Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería en Informática

PROYECTO FIN DE CARRERA

Evolución de distancias para clasificadores basados en prototipos

Autor: Christian Felipe Álvarez

Tutores: Ricardo Aler Mur

José María Valls Ferrán

Mayo, 2010

Agradecimientos

Siempre que se completa una etapa importante de la vida se pasa por un periodo agridulce: uno se encuentra ante un nuevo abanico de posibilidades al que mira con una mezcla de alegría, fascinación y recelo; a la vez que comienza a echar de menos todo aquello que deja atrás. Este proyecto representa el final de lo que seguramente sea una de las mejores etapas de mi vida y quiero mostrar mis agradecimientos a todas las personas que, de una forma u otra, me han acompañado en ella.

A mis padres, por todo el apoyo que me han dado estos años, enseñándome a aprender a vivir mi vida y brindándome con ilusión oportunidades y medios con los que ellos no contaron.

A mi hermana, por siempre haberme intentado ayudar, colaborando en lo que ha podido, con consejos o simplemente escuchando mis movidas y paranoias, pero siempre con la mejor de sus intenciones.

A José María Valls y Ricardo Aler, por su tutela en este proyecto, por haberse portado tan bien conmigo durante el desarrollo del mismo y por todas sus recomendaciones y apoyo.

A todos mis compañeros del Laboratorio del Departamento de Informática: técnicos, becarios, becarios que se fueron y volvieron como técnicos, visitantes frecuentes, visitantes ocasionales... que siempre han estado ahí para echarme una mano cuando ha hecho falta. En este lugar, donde siempre se ha dicho que *no debería de pasar nada*, he aprendido más de lo que hubiera podido esperar y vivido momentos que no se me van a olvidar. Dentro del laboratorio, quiero agradecer especialmente a Óscar su confianza, ayuda y consejos.

A todos los compañeros que han sufrido conmigo tantas prácticas insufribles y con los que he trabajado para superar las asignaturas. La primera práctica de programación que terminamos a las tantas en mi casa, los tanques que después de días evolucionando no disparaban, la inmersión voluntaria en el mundo de los fractales, las noches que pasábamos informándonos sobre cannabis para hacer un sistema experto, y un largo etcétera, que recordaré siempre.

Por último, no podía dejar de agradecer el haber estado ahí cuando lo he necesitado y el haberme dado tantos momentos felices a todos mis amigos, tanto los que me llevo de la

universidad como los que ya me acompañaban de antes. A Miguel, por su gran amistad y todo lo que hemos vivido juntos; a Adrián por esas conversaciones con las que arreglamos el mundo, y a su pareja Diana, con la que siempre resulta fácil hablar y por esa alegría suya; a Javi, la primera persona a la que conocí en la universidad y que, desde entonces, siempre ha estado ahí para apoyarme; a Iván, por sus charlas y *frikadas*, y el gran apoyo que ha sido para mí a la hora de hacer este proyecto, siendo el único con el que podía hablar sobre él en profundidad; a Lisardo, que siempre me ha dado una dosis de buen rollo y con el que he aprendido tantas cosas; a José, su *vaya horitas* y todas esas conversaciones; a Óscar y a Juanma, por lo bien que lo paso con ellos, también por las risas, bastedades y lo buena gente que son; A Sergio, por cómo se ha preocupado siempre y todas las charlas que hemos tenido; a Lidia por su amistad y ese viaje a Londres del que trajimos tantos recuerdos; a Laura por esa personalidad suya, por saber escuchar y por las veces que se deja picar; a Alberto por cómo te explica las cosas y lo bien que se porta; A Patri y sus largas conversaciones telefónicas; a Carmen, por siempre obsequiarme con una sonrisa, a Rober por lo buen tío que es aunque no le guste mucho moverse, a Gon con el que se puede dialogar y discutir cuando no tiene a su fuerte apoyo y a Carlos, su fuerte apoyo; a Antonio, por las charlas que teníamos mientras comíamos y la práctica de ficheros se quedaba ejecutando tranquilamente; y a Nacho, por sus peculiaridades y siempre tratar de echar una mano.

A todos vosotros, gracias de todo corazón.

Christian Felipe Álvarez.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Mediante un algoritmo de agrupamiento o técnica de *clustering* se trata de distribuir una serie de datos de un conjunto en subconjuntos, de forma que los datos pertenecientes a cada subconjunto tengan algún tipo de similitud. Por otro lado, utilizando técnicas de clasificación, se pretende determinar a qué clase pertenece una muestra recogida, haciendo uso del conocimiento adquirido al procesar una serie de observaciones que habían sido previamente clasificadas de forma correcta. En general, los algoritmos de clasificación también utilizan el concepto de similitud para determinar la clase de los datos.

Ese concepto de similitud suele entenderse como la distancia entre puntos (correspondientes a la representación de muestras de datos). Al utilizar la Distancia Euclídea, se asume implícitamente que todos los atributos de las muestras tienen la misma relevancia a la hora de medir la similitud. Esto no es lo más adecuado para todos los dominios, por lo que se han propuesto métodos en los que se utilizan otros tipos funciones de distancia, como la distancia de Mahalanobis [Haasdonk, 2008], [Wang, 2004] o la distancia de Chebyshev [Ozer, 2008].

Los métodos de agrupamiento no sirven para abordar problemas de clasificación, se utilizan para descubrir similitudes entre los datos, encontrando posibles clases; pero no permiten predecir la clase de un nuevo dato basándose en un conjunto de ejemplos conocidos y clasificados. Sin embargo, pueden construirse métodos de clasificación basados en técnicas de agrupamiento. Un ejemplo bien conocido de esta forma de abordar los problemas de clasificación es el método de Redes de Cuantización Vectorial [Kohonen, 1986], en inglés, *Learning Vector Quantization, LVQ*, basado en los Mapas Auto-organizativos de Kohonen [Kohonen, 1995] (*Kohonen's Feature Map*).

Otro algoritmo de agrupamiento que es posible utilizar para conseguir un método de

clasificación es el K-Medias (*K-Means*). Esta técnica se basa en la distribución de k patrones o prototipos, de forma que cada uno de ellos será el centro de una clase. En cada iteración los datos se asignan a los patrones según su proximidad y los patrones se mueven hacia el centro de masas de los ejemplos asignados. Por lo tanto, tras la ejecución de K-Medias se obtienen k prototipos representativos de k subconjuntos de datos con características similares.

Si los patrones obtenidos corresponden a clases determinadas, se pueden clasificar nuevos ejemplos comparándolos con dichos patrones, estableciendo una correspondencia entre cada dato y el prototipo más cercano. De esta forma se dispone que el nuevo dato pertenece a la clase que le corresponde al patrón más próximo.

El objetivo de este proyecto es encontrar una medida de distancia que mejore los resultados del método de clasificación basado en K-Medias que se ha descrito anteriormente, respecto a los que se obtienen utilizando la Distancia Euclídea. Para ello se plantea un método en el que la medida de distancia se pueda adaptar a las características del dominio y a las de la técnica de clasificación que se utilice.

La distancia de Mahalanobis hace uso de la matriz de covarianza del conjunto de datos para obtener la medida de distancia, teniendo así en cuenta las correlaciones entre los datos. Por ello, la distancia de Mahalanobis está más adaptada a los datos del problema que la Distancia Euclídea, pero, a pesar de ello, no se tienen en cuenta las características del método de clasificación utilizado. Por este motivo, en lugar de utilizar la distancia de Mahalanobis, se utiliza la distancia Euclídea Generalizada, que se diferencia de esta última en que, en lugar de utilizar la matriz de covarianza, se utiliza una matriz que establece la ponderación de la función de distancia.

Esta matriz debe hallarse de forma que se adapte al dominio y al algoritmo de clasificación. Es decir, la matriz debe optimizarse para que el error cometido en la clasificación sea el menor posible. El proceso de la optimización de la matriz, y por tanto, la optimización de la función de distancia, se realiza utilizando técnicas de computación evolutiva, concretamente estrategias evolutivas.

Este trabajo incluye el desarrollo de una aplicación que implemente el algoritmo de clasificación descrito y la experimentación con la herramienta desarrollada en distintos dominios.

Capítulo 2

Contexto

2.1. El algoritmo K-Medias

Stuart Lloyd propuso el algoritmo K-Medias en 1957, como una técnica de modulación por impulsos codificados (*PCM*, por sus siglas en inglés, *Pulse-Code Modulation*) [Lloyd, 1982], pero su trabajo no se publicó hasta 1982. En 1967 James MacQueen [MacQueen, 1967] utilizó el término K-Medias por primera vez para referirse a un método para particionar un espacio de datos en subconjuntos.

2.1.1. Descripción del algoritmo

K-Medias es una técnica de agrupamiento, pues permite dividir n objetos en k clases. El agrupamiento o *clustering* es útil por distintos motivos. En primer lugar, se trata de una herramienta que facilita la comunicación: si contamos con una serie de objetos (que presenten algunas características similares) agrupados, tenemos la posibilidad de asignar una identidad a ese grupo, esto permite que nos refiramos a cualquiera de los elementos de dicho grupo utilizando un nombre común. Por otra parte, el hecho de agrupar los elementos en distintos grupos o clases nos facilita hacer predicciones: cuando encontramos un nuevo elemento y establecemos que pertenece a una cierta clase conocida, esperamos que se comporte de una forma similar que el resto de los elementos de dicha clase.

Dentro del aprendizaje automático, el algoritmo K-Medias es un método de aprendizaje no supervisado porque nos permite obtener información sobre los datos sin que aportemos un conocimiento previo sobre los mismos.

Para agrupar n datos, que denotaremos con los vectores x_i, con $i = \{1, 2, ..., n\}$, en k grupos, utilizaremos k vectores llamados prototipos o centroides c_i, con $i = \{1, 2, ..., k\}$. Se asume que los vectores son reales y que contamos con alguna medida de distancia $d(x, y)$. Por ejemplo, se puede utilizar la Distancia Euclídea.

El algoritmo consiste en la iteración de dos pasos: asignar cada ejemplo al centroide más cercano y actualizar el valor de los centroides como la media de los datos que tenga asignados. De forma más concreta, la **especificación del algoritmo K-Medias** es la siguiente:

Paso 0: inicialización. Se inicializan los centroides c_j con valores aleatorios.

Paso 1: asignación. Se asigna cada ejemplo x_i al centroide c_j más cercano. Representamos esta asignación mediante conjuntos $\hat{C}$, de forma que $\hat{C}_j$ contiene los datos para los que c_j es el centroide más cercano.

$$\hat{C}_j = \{x_i \ : \ d(x_i, c_j) \ \leq \ d(x_i, c_k) \ \forall \ k \ \neq \ j\}$$

donde la función $d(x, c)$ es una medida de distancia.

Paso 2: actualización. Se actualiza el valor de cada centroide c_j como la media de los ejemplos que tiene asignados.

$$c'_j = \frac{1}{|\hat{C}_j|} \sum_{x_i \in \hat{C}_j} x_i$$

Se repiten los pasos 1 y 2 hasta que el valor de los centroides se estabilice.

$$c'_j \approx c_j \ \forall \ j$$

Se ha definido un pequeño conjunto de datos para mostrar el funcionamiento del algoritmo. En la figura 2.1, se han representado los ejemplos como círculos y los centroides como cuadrados.

En primer lugar, los centroides se inicializan aleatoriamente, para a continuación, comenzar la primera iteración del algoritmo. Los centroides y los datos que corresponden a cada uno de ellos se han representado del mismo color, pudiéndose así apreciar qué ejemplos son asignados a cada centroide.

Se puede observar que en la primera iteración, se asignan los datos a los centroides más cercanos y se mueve el centroide al punto medio de éstos. Se repite el mismo proceso en

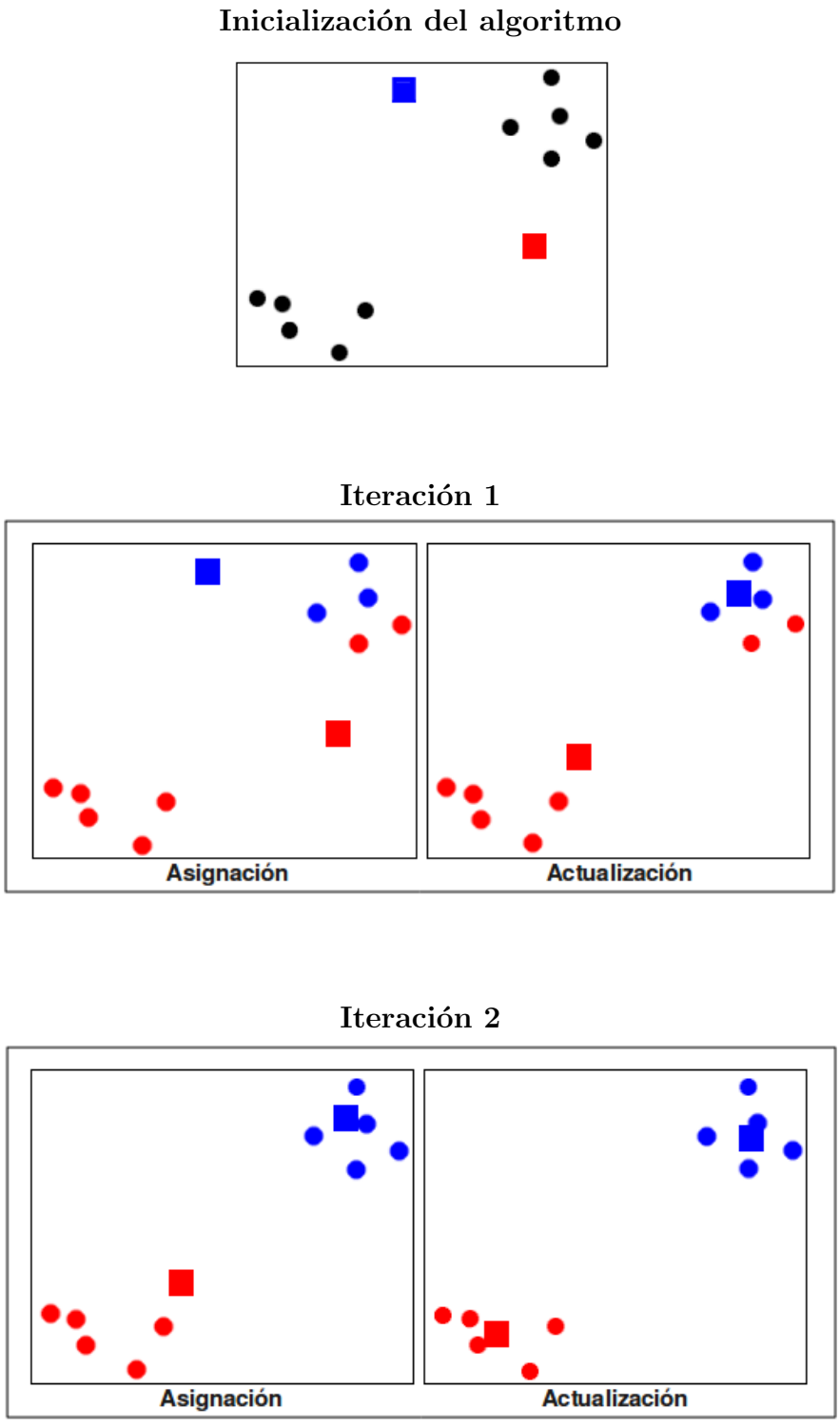


Figura 2.1: Ejemplo de ejecución del algoritmo K-Medias.

la segunda iteración, y en la tercera iteración, los conjuntos de datos asignados no varían. Esto implica que los valores de los centroides se mantienen, convergiendo el algoritmo en tres iteraciones.

2.1.2. Inicialización del algoritmo

Uno de los principales problemas del algoritmo K-medias es que depende, en gran medida, de la inicialización de los centroides. Típicamente se inicializan aleatoriamente, pero determinadas inicializaciones provocarán que no se obtenga un agrupamiento adecuado, es decir, provocarán que se converja a soluciones subóptimas. Por otra parte, aunque no se caiga en mínimos locales, la situación inicial de los centroides también va a influir en el número de iteraciones necesario para alcanzar una solución.

Al utilizar valores arbitrarios como valores iniciales para los centroides, no se establece ningún control sobre la inicialización del algoritmo, por lo que es posible que este tipo de problemas surjan. Para evitarlo, existen distintos métodos para inicializar los centroides con los que se pretende mejorar el algoritmo en lo que respecta a la velocidad, y a la calidad de la solución.

Una posibilidad es elegir como centroide **el punto más alejado** del resto de centroides ya asignados, los valores iniciales de los centroides se elegirían siguiendo los siguientes pasos:

Paso 1. Se elije el primer centroide aleatoriamente dentro del conjunto de datos y se añade al conjunto C de centroides.

Paso 2. Se calcula la máxima distancia de D_i para cada ejemplo x_i del conjunto de datos.

$$D_i = \max_{c_j \in C}\{d(x_i, c_j)\}$$

Paso 3. Se elije como centroide el punto x_i cuya distancia sea máxima y se añade al conjunto de centroides C.

Paso 4. Se repiten los pasos 2 y 3 hasta que se hayan calculado los k centroides, es decir, hasta que $|C| = k$.

El problema de este método de el punto más lejano es que es muy sensible a ejemplos atípicos, tal como puede observarse en la figura 2.2.

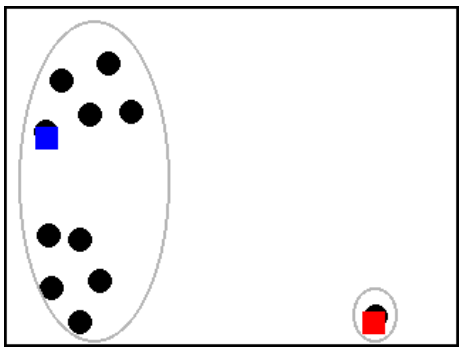


Figura 2.2: Problema al inicializar K-Medias mediante *el punto más lejano*.

Si el punto azul corresponde al primer centroide, el segundo que se asigna es el punto rojo, que es el ejemplo más lejano. Esto provoca que, tras la ejecución de K-Medias, los grupos asignados correspondan con los elipses marcados.

Una variante del K-Medias, conocida como K-Medias++ (originalmente, *K-Means++*) [Arthur, 2007], también se basa en mejorar la inicialización de los centroides.

El algoritmo K-Medias original utiliza valores aleatorios como centroides iniciales, por lo que es posible que se den los problemas anteriormente citados, y utilizando el método del punto más lejano, existe una fuerte sensibilidad a los datos atípicos.

Mediante el algoritmo K-Medias++ se pretende reducir estos problemas combinando ambos métodos. Se mantiene la aleatoriedad para que el método no sea sensible a valores atípicos, pero se tiene en cuenta la distancia a los centroides ya asignados, no siendo la inicialización totalmente aleatoria.

Esto se consigue introduciendo una función de probabilidad, de forma que los puntos más alejados de los centroides ya elegidos son más susceptibles de ser asignados como nuevos centroides. Por lo tanto, la **especificación del algoritmo K-Medias++** es la siguiente.

Definimos:

$X = \{x_1, x_2, \ldots, x_n\}$ como el conjunto de datos.

$C = \{c_1, c_2, \ldots, c_k\}$ como el conjunto de centroides iniciales ya elegidos.

$D(x)$ como una función que denota la distancia entre el punto x y el centroide más

cercano elegido.

$$D(x) = \min_{c_i \in C}\{d(x, c_i)\}$$

Paso 1. Se elige un centroide inicial c_1 aleatoriamente de X.

Paso 2. Se elige el siguiente centroide $c_i = x'$, con probabilidad $\dfrac{D(x')^2}{\sum_{x \in X} D(x)^2}$.

Paso 3. Se repite el paso 2 hasta haber elegido los k centroides, $|C| = k$.

Paso 4. Se ejecuta el algoritmo K-Medias estándar utilizando como centroides iniciales los del conjunto C.

2.2. Funciones de distancia

La distancia permite cuantificar el concepto de similitud entre dos elementos respecto a una serie de características que ambos presentan; establece la lejanía y proximidad entre dos objetos.

Formalmente, una función de distancia, o métrica, en un conjunto X es una función

$$d \ : \ X \ \times \ X \ \rightarrow \ \mathbb{R}$$

que cumple las siguiente propiedades [Johnsonbaugh, 2002]:

1. $d(x, y) \ \geq \ 0 \ \ \forall \, x, y \in X$ (No negatividad).
2. $d(x, y) \ = \ d(y, x) \ \ \forall \, x, y \in X$ (Simetría).
3. $d(x, y) \ = \ 0 \ \Leftrightarrow \ x \ = \ y \ \ \forall \, x, y \in X$ (Definición positiva).
4. $d(x, z) \ \leq \ d(x, y) \ + \ d(y, z) \ \ \forall \, x, y, z \in X$ (Desigualdad triangular).

Hay problemas en los que puede ser interesante *relajar* algunas de estas condiciones para ciertos conjuntos de datos [Atkeson, 1997] (problemas de regresión, clasificación, etc.). Teniendo esto en cuenta, se pueden definir funciones de distancia de distintas formas:

- **Funciones de distancia globales**: se utiliza la misma función de distancia para todos los datos del dominio.
- **Funciones de distancia locales basadas en consultas**: la función de distancia se modifica mediante algún proceso de optimización para mejorar los resultados del proceso para el que se esté utilizando. La función de distancia se adapta a las técnicas empleadas y a los datos con los que se emplean.
- **Funciones de distancia locales basadas en puntos**: se asocia cada dato con una función de distancia, de este modo, para cada punto puede ser diferente la forma de determinar la distancia.

2.2.1. Distancia Euclídea

En el conjunto $\mathbb{R}^n$, definido por vectores reales de dimensión n, la función de distancia más común es la Distancia Euclídea. La expresión 2.1[1] corresponde con dicha función de distancia.

$$d(x,y) \;=\; \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2} \;=\; \sqrt{(x-y)(x-y)^T} \tag{2.1}$$

Se trata de la función de distancia más básica, y que nos da la idea intuitiva de distancia como longitud. En el espacio $\mathbb{R}^2$ corresponde con la longitud del segmento de recta que une dos puntos. En este mismo espacio, el conjunto de los puntos equidistantes a un punto x, $C = \{c_i \;:\; d(x,c_i) = r\}$, forma una circunferencia, de radio r con centro en x.

2.2.2. Distancia de Mahalanobis

La Distancia de Mahalanobis [Mahalanobis, 1936] introduce la correlación entre los datos en el cálculo de la distancia, por medio de la matriz de covarianza.

Si x e y pertenecen al conjunto $X \in \mathbb{R}^n$, y Σ es la matriz de covarianza de los vectores del conjunto X, entonces 2.2 es expresión de la función de Distancia de Mahalanobis.

Realmente, la Distancia de Mahalanobis es un tipo de distancia ponderada, en la que se establece la ponderación según la matriz de covarianza de un conjunto de datos. La matriz de covarianza que caracteriza a la distancia corresponde al elipsoide que mejor representa la distribución de probabilidad del conjunto de datos X.

$$d_\Sigma(x,y) \;=\; \sqrt{(x-y)\Sigma^{-1}(x-y)^T} \tag{2.2}$$

[1] Se considera que en este tipo de expresiones los vectores como x o y son vectores fila.

2.2.3. Distancia Euclídea ponderada

La Distancia Euclídea Ponderada (del inglés *weighted Euclidean distance* o *diagonally weighted Euclidean distance*) está caracterizada por una matriz diagonal que contiene los factores de escala o ponderación para cada una de las dimensiones de los datos.

$$d_W(x,y) \;=\; \sqrt{\sum_{i=0}^{n}(W_{di,i}(x_i - y_i))^2} \;=\; \sqrt{(x-y)W_d\, W_d^T(x-y)^T} \tag{2.3}$$

En este caso, al producirse un escalado de las dimensiones de los datos cuando se hace el cálculo de la distancia, los datos equidistantes a un punto dado, generan una elipse en dimensión n.

La fórmula 2.3 corresponde a la función de Distancia Euclídea Ponderada, en la que la matriz $W_d \in \mathbb{R}^{n\times n}$ es diagonal, y para la que $W_{di,i}$ es el factor de escala para la dimensión i de los vectores.

Si $W = W_d W_d^T = W_d^2$, al ser W_d una matriz diagonal, la ecuación 2.3 puede expresarse como 2.4.

$$d_W(x,y) \;=\; \sqrt{\sum_{i=0}^{n} W_{i,i}(x_i - y_i)^2} \;=\; \sqrt{(x-y)W(x-y)^T} \tag{2.4}$$

2.2.4. Distancia Euclídea Generalizada

La Distancia Euclídea, la Distancia de Mahalanobis y la Distancia Euclídea Ponderada, pueden verse como casos concretos Distancia Euclídea Generalizada o *GED* (por sus siglas en inglés *Generalized Euclidean Distance*, también conocida como *fully weighted Euclidean distance*). Esta función de distancia está caracterizada por una matriz $M_d \in \mathbb{R}^{n\times n}$. La ecuación 2.5 corresponde con la función de la Distancia Euclídea Generalizada.

$$d_M(x,y) \;=\; \sqrt{(x-y)M_d\, M_d^T(x-y)^T} \tag{2.5}$$

Siendo $M = M_d\, M_d^T$, que por definición es una matriz simétrica, la fórmula 2.5 se puede

expresar como 2.6.

$$d_M(x,y) \;=\; \sqrt{(x-y)M(x-y)^T} \tag{2.6}$$

Para que la *GED* cumpla las propiedades que caracterizan a las funciones de distancia, es necesario que la matriz M sea definida positiva.

A continuación, se muestra cómo las funciones de distancia descritas anteriormente son casos concretos de la Distancia Euclídea Generalizada:

- **Distancia Euclídea**: $d(x,y) = d_M(x,y)$, con $M = I$.
- **Distancia de Mahalanobis**: $d_\Sigma(x,y) = d_M(x,y)$, siendo M la matriz de covarianza inversa.
- **Distancia Euclídea ponderada**: $d_W(x,y) = d_M(x,y)$, siendo M una matriz diagonal.

En las figuras 2.3, 2.4 y 2.4, se han representado varios conjuntos de puntos equidistantes (según la función de distancia indicada) al centro $(0,0)$, en el espacio $\mathbb{R}^2$.

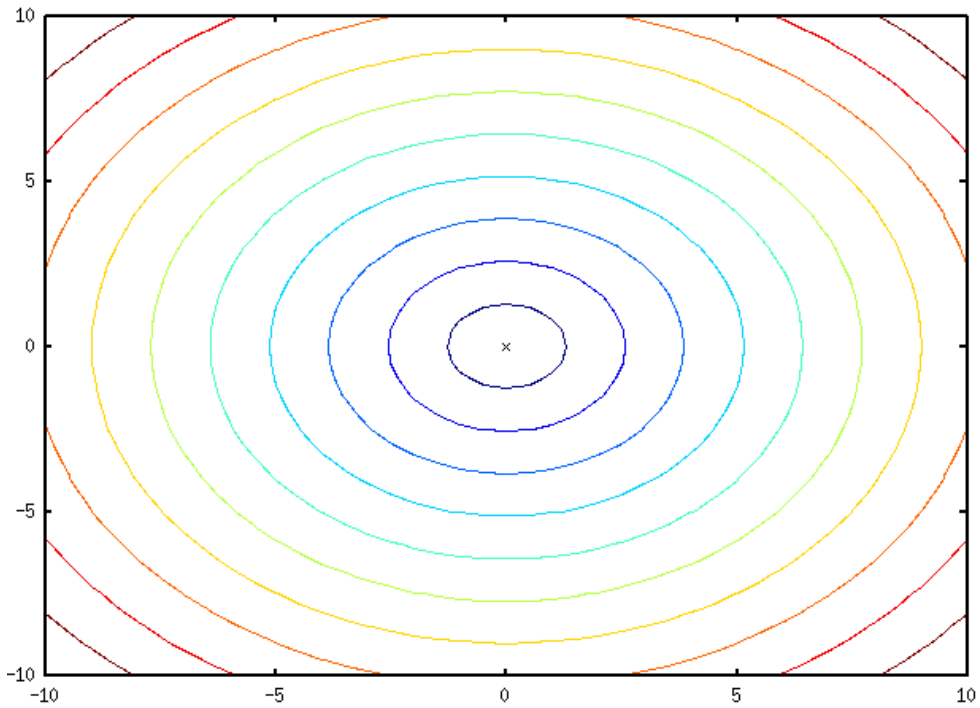


Figura 2.3: Representación de la función de la Distancia Euclídea.

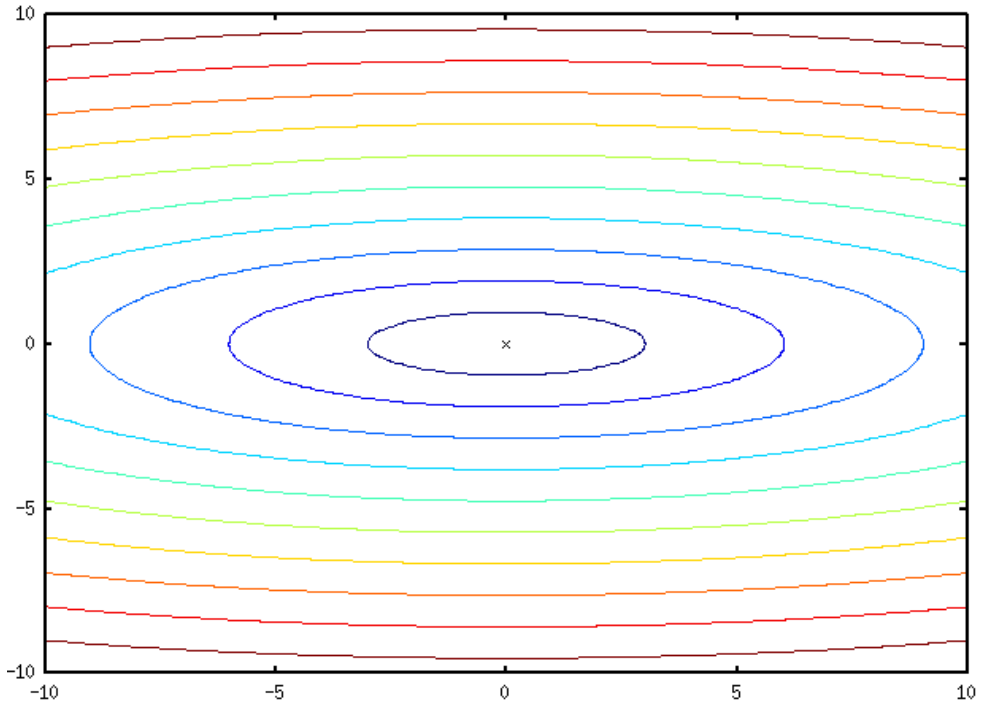


Figura 2.4: Representación de una función de Distancia Ponderada, con $W = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$.

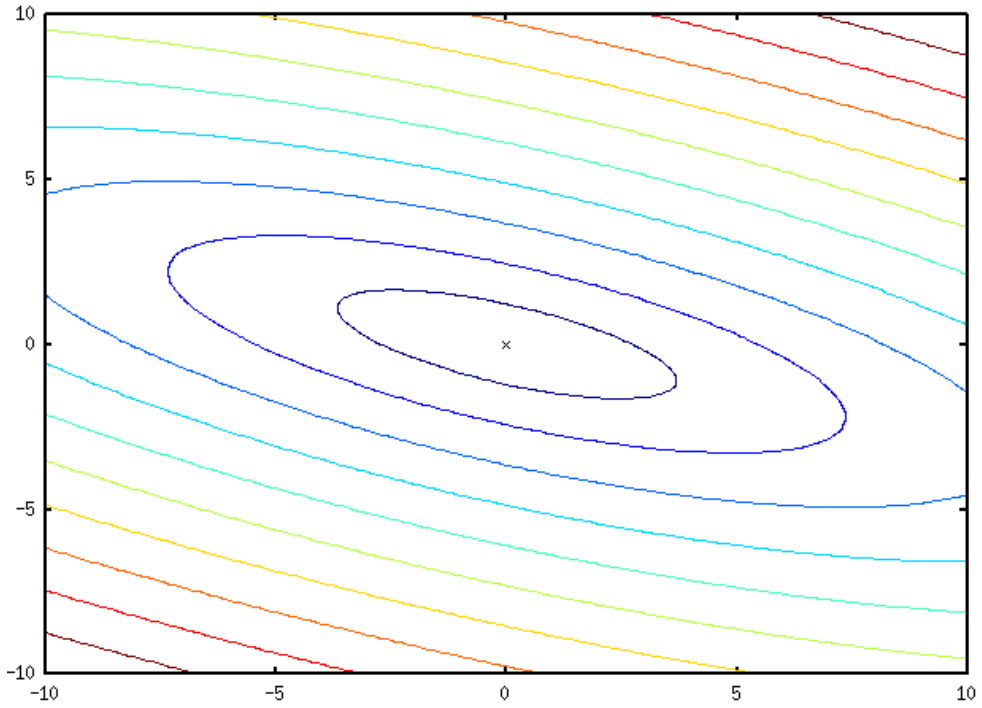


Figura 2.5: Representación de una función de Distancia Generalizada, con $M = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 2 \end{bmatrix}$.

2.2.5. Distancia Euclídea Generalizada y Proyecciones

Una proyección es una transformación lineal idempotente que hace corresponder un punto x cualquiera de un espacio vectorial al subespacio imagen de la transformación [Meyer, 2000].

Se puede caracterizar el operador de proyección utilizando la matriz P correspondiente al espacio vectorial imagen. Entonces, calculamos $x^{'}$, la proyección del vector x en el espacio imagen de P, mediante la ecuación 2.7.

$$x^{'} \;=\; x \cdot P \tag{2.7}$$

Si la matriz $P = M_d$, es equivalente calcular la Distancia Euclídea de los vectores x e y proyectados $d(x\,M_d, y\,M_d)$, que la Distancia Euclídea Generalizada $d_M(x, y)$, tal como se muestra en la expresión 2.8.

$$\begin{aligned} d(xM_d, yM_d) = \sqrt{(xM_d - yM_d)(xM_d - yM_d)^T} = \\ \sqrt{(xM_d - yM_d)((xM_d)^T - (yM_d)^T)} = \\ \sqrt{(xM_d - yM_d)(M_d^T x^T - M_d^T y^T)} = \\ \sqrt{(x - y)\; M_d\; M_d^T\; (x^T - y^T)} = \\ \sqrt{(x - y)\; M_d\; M_d^T\; (x - y)^T} = d_M(x, y) \end{aligned} \tag{2.8}$$

2.3. Computación evolutiva

En las décadas de 1950 y 1960, varios científicos comenzaron a estudiar el comportamiento de los sistemas evolutivos en la naturaleza para tratar de aplicar la evolución a problemas de ingeniería. La idea era evolucionar un conjunto de posibles soluciones a un problema utilizando operadores inspirados en la variabilidad genética y el proceso selección natural postulado por Darwin [Darwin, 1921].

En este periodo varios investigadores trabajaron en el desarrollo de métodos de optimización y aprendizaje automático inspirados en la evolución. En los años sesenta Rechenberg introdujo el concepto de **estrategias evolutivas** [Rechenberg, 1973] como método de optimización de valores reales, este método sería desarrollada más adelante por Schwefel [Schwefel, 1975]. Por otro lado, Fogel, Owen y Walsh presentaron una técnica en la que los elementos del conjunto de posibles soluciones venían representados como autómatas de estado finito, la **programación genética** [Fogel, 1964]. En la misma época John Holland inventó los **algoritmos genéticos** al estudiar cómo se da en la naturaleza la adaptación y tratar de llevar ese concepto a los sistemas de computación [Holland, 1975].

2.3.1. Motivación de la computación evolutiva

Existen ciertos problemas para los que es preciso dar con una solución aceptable entre un gran número de posibles soluciones, para gran parte de estos problemas no se cuenta con un método eficiente de búsqueda que permita hallar una solución con los recursos computacionales disponibles.

También hay casos en los que los programas deben ser adaptativos: cuando es necesario que se ejecuten correctamente en entornos variables y tienen que resolver correctamente las situaciones que el propio entorno o los usuarios requieran. Otra característica deseable en los programas que se precisarían para resolver algunos problemas es la innovación, necesaria cuando se intentan descubrir nuevas técnicas para resolver tareas computacionales o para realizar otros tipos de descubrimientos. Por último, algunos problemas precisan soluciones complejas, demasiado difíciles de codificar para las personas.

El proceso de evolución que se aprecia en la naturaleza, puede verse como una fuente de inspiración para resolver este tipo de problemas. Se pueden considerar las posibles

secuencias genéticas de los organismos como posibles soluciones al problema de la supervivencia en el entorno, y como soluciones adecuadas a aquellos individuos que son capaces de sobrevivir por estar suficientemente adaptados a dicho entorno.

Estos entornos cambian constantemente según los organismos evolucionan, por lo que la evolución se desenvuelve en un conjunto continuamente cambiante de posibilidades. Es fácil apreciar características innovadoras en la naturaleza, un ejemplo puede encontrarse en las relaciones entre parásitos y hospedadores, como el caso de una especie de cuclillo que pone los huevos en nidos de aves de otras especies para que éstas los incuben [Dawkins, 2002].

Además, la evolución funciona según unas reglas bastante simples: las especies evolucionan mediante variaciones aleatorias, producidas mediante mutaciones, cruces, etc.; seguidas de selección natural, mediante la que se tiende a que el más apto sobreviva y se reproduzca propagando su material genético. En cierto modo, toda la diversidad y complejidad que encontramos en la naturaleza ha surgido mediante estas simples reglas.

2.3.2. Términos biológicos utilizados en computación evolutiva

En el contexto de la computación evolutiva se suelen emplear algunos términos biológicos, estos términos se emplean para establecer analogías entre los algoritmos de inspiración biológica y los propios procesos biológicos en los que se inspiran. Cabe mencionar, que los conceptos que representan estos términos en el ámbito de la computación evolutiva son mucho más simples que los correspondientes en el ámbito de la biología.

En biología, los **cromosomas** son pequeños cuerpos contenidos en las células y en los que se organiza el ADN, en este material está alojada la información genética de los organismos vivos. De forma conceptual, los cromosomas pueden dividirse en **genes**. Los genes están vinculados al desarrollo o al funcionamiento de las distintas funciones fisiológicas; son considerados como la unidad de almacenamiento de información genética y como lo que cada ser vivo transmite a sus descendientes que provoca que prole y progenitores tengan características comunes. Las distintas posibilidades de configuración de los genes reciben el nombre de **alelos**, por ejemplo, si la característica vinculada a un determinado gen es el color del cabello, los alelos serían castaño, moreno, rubio, etc.

Las células pueden contener varios cromosomas, se llama **genoma** al conjunto de todo

el material genético contenido en las células de los organismos. Al conjunto de genes en forma de ADN que definen a un individuo se le denomina **genotipo**, mientras que al conjunto de características vinculadas a dichos genes se le llama **fenotipo**.

Si los cromosomas se organizan formando parejas estamos hablado de organismos *diploides*, en caso contrario se denominan organismos *haploides*. La mayor parte de los seres vivos que se reproduce mediante reproducción sexual son diploides. Durante la reproducción sexual se produce el **cruce** o recombinación (en inglés, *crossover*). En los organismos diploides el cruce consiste en lo siguiente: en cada uno de los progenitores, los genes se intercambian en cada par de cromosomas formando un *gameto* (un cromosoma no emparejado); entonces, los gametos de ambos progenitores se combinan formando un único cromosoma completo. En el caso de los organismos haploides, directamente se combinan los genes de los cromosomas singulares de los progenitores.

En ocasiones, al copiarse los genes durante la reproducción, se producen ligeras variaciones en el ADN, produciéndose cambios en el genotipo y el fenotipo. Este fenómeno es conocido como **mutación**. Por otra parte, el factor de **adecuación** (*fitness*) de los organismos vivos viene determinado por las posibilidades que éstos tienen de vivir suficiente tiempo como para reproducirse y que sus genes continúen propagándose.

En el contexto de la computación evolutiva, los *cromosomas* se suelen representar como cadenas (de bits, de números reales, o con otras representaciones), que pueden dividirse en lo que se consideran *genes*. De modo que los alelos vendrían determinados por los posibles valores que pudieran tomar los genes (siendo bits, se trataría de los valores 0 ó 1).

El *cruce* consiste en la combinación de los cromosomas de dos individuos y la *mutación* consiste en cambiar aleatoriamente el valor de algunos de sus genes. El *valor de adecuación* se determina mediante algún tipo de función que establece lo adecuada que es la solución al problema tratado que corresponde al individuo.

El significado de estos términos variará según la técnica computacional evolutiva con la que se esté trabajando, pero en todos los casos se mantiene el significado esencial análogo al significado biológico que ha servido para inspirar este tipo de algoritmos.

2.3.3. Operadores evolutivos básicos

Los operadores más comunes utilizados en las implementaciones de algoritmos evolutivos son la *selección*, el *cruce* y la *mutación*, aunque no es necesario que los tres operadores estén presentes en todo algoritmo de computación evolutiva, y la implementación de dichos operadores puede diferir, en mayor o menor medida, entre distintas técnicas de computación evolutiva. A continuación se ofrece una descripción breve de estos operadores.

Gran parte de los algoritmos evolutivos constan de una población de individuos que están caracterizados por un cromosoma. Los individuos se reproducen, y el operador de **selección** permite determinar qué individuos de la población son los que van a reproducirse. Las *estrategias de reemplazo* están relacionadas con la selección, consisten en establecer qué individuos se mantienen entre los descendientes y los progenitores. Estos operadores deben tener en cuenta el valor de adecuación de los individuos y permitir que la búsqueda en el espacio de soluciones esté bien dirigida. Entre los operadores de selección más utilizados encontramos los siguientes:

- **Selección por trucamiento**: consiste en que sean los individuos con mayor valor de adecuación los que se reproduzcan entre si. Este método es muy simple y es muy posible que se caiga en mínimos locales.
- **Método de la ruleta** [Holland, 1975]: se trata de un método en el que la selección se hace de forma proporcional al valor de adecuación, pero que también permite reproducirse a los peores individuos. La probabilidad de que un individuo con valor de adecuación f_i en una población de n individuos se reproduzca viene dada por la expresión 2.9.

  $$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2.9}$$

  Conceptualmente, el método consiste en asignar la parte de una ruleta proporcional a su valor de adecuación a cada individuo, y a continuación, lanzar una bola, siendo el individuo en cuya parte de la ruleta la bola se pare el elegido para reproducirse. Habría que repetir el proceso tantas veces como individuos se quieran seleccionar para la reproducción. Este método presenta problemas cuando pocos individuos de la población tienen un valor de adecuación muy alto, pues prácticamente siempre serán ellos los elegidos, dejándose entonces de mantenerse la variabilidad genética.
- **Selección por torneo** [Miller, 1995]: consiste en seleccionar al individuo con

mejor adecuación de un pequeño subconjunto de la población elegido al azar. Es necesario hacer tantos torneos como individuos se pretenda seleccionar, y hay que establecer el tamaño del subconjunto de individuos. Este método permite evitar, hasta cierto punto, la convergencia prematura; la selección aleatoria del subconjunto de individuos que compiten en el torneo es una forma de mantener la variabilidad genética.

La reproducción de los individuos se efectúa a través del operador de **cruce**. El cruce consiste en *mezclar* los cromosomas de un individuo para dar lugar a uno nuevo. Existen distintos tipos de cruce o *sobrecruzamiento*, los más populares [Wook-Ahn, 2006] son el cruce simple en un punto (*one-point crossover*) y el cruce uniforme (*uniform crossover*). El cruce simple consiste en elegir un punto de cruce que divide los cromosomas de los progenitores en dos secuencias de genes e intercambiar estas secuencias para generar dos sucesores, se incluye un ejemplo de este tipo de cruce en la figura 2.6. Del mismo modo se puede utilizar el *cruce multipunto*, si se elige más de un punto de cruce y se intercambian varias secuencias de genes. El cruce uniforme consiste en intercambiar cada uno de los genes del cromosoma con una probabilidad del $50\,\%$.

Al generar los nuevos individuos mediante cruce pueden surgir ciertos problemas. Es posible que ocurra que en una población ninguno de los individuos presente un determinado valor para ciertos genes, como los individuos se reproducen entre sí, estos valores nunca se alcanzan, dicho de otro modo, se producen pérdidas de alelos. Esto implica que la diversidad genética se pierde rápidamente, y consecuentemente el algoritmo evolutivo converge demasiado pronto. Para intentar solventar este tipo de problemas se introduce otro operador: la **mutación**. Este operador consiste en alterar de forma aleatoria algunos de los genes de los individuos, con cierta probabilidad. Así se introduce aleatoriedad en la búsqueda, evitándose la pérdida de alelos y que la población pierda variabilidad prematuramente.

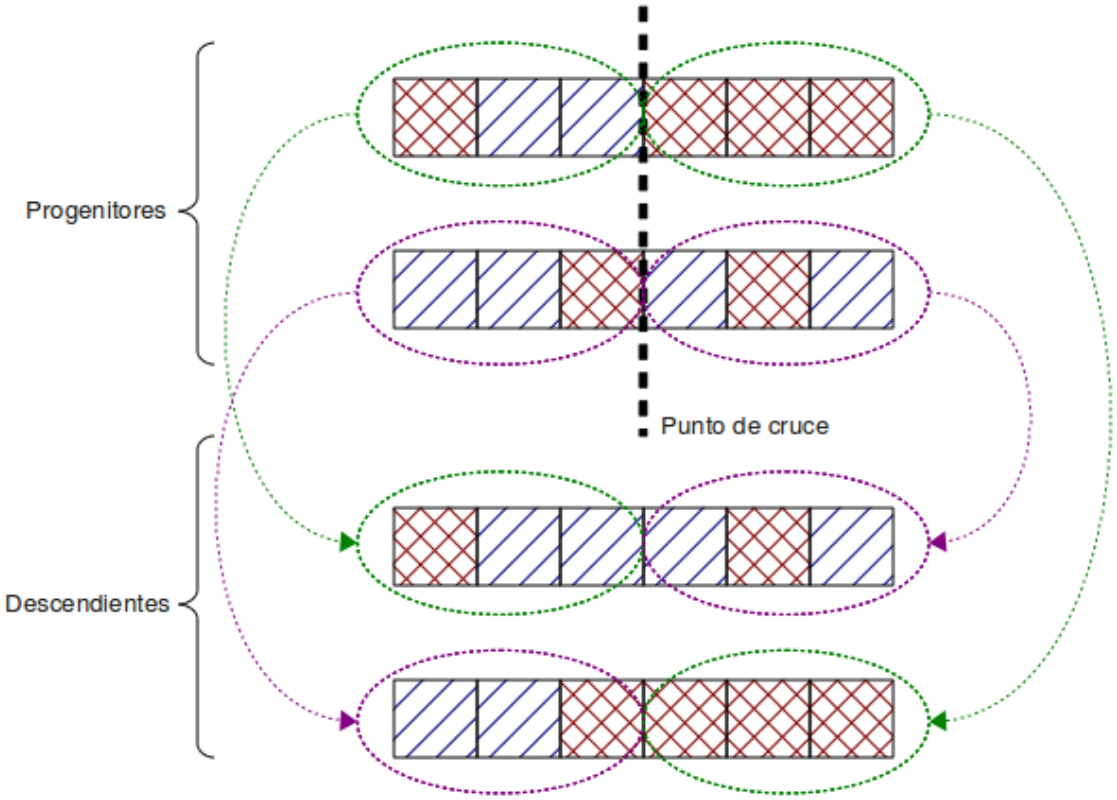


Figura 2.6: Ejemplo de cruce simple en un punto.

2.4. Estrategias evolutivas

Las estrategias evolutivas [Bäck, 1991] son un tipo de técnicas de computación evolutiva en las que se codifican y tratan de optimizarse valores reales. Rechenberg [Rechenberg, 1973] propuso este término y llevó a cabo los primeros trabajos al respecto en la Universidad Técnica de Berlín en 1964. La idea era imitar los principios orgánicos para proponer soluciones en el campo de la optimización de parámetros.

2.4.1. Estrategias evolutivas (1+1)

Las primeras estrategias evolutivas propuestas fueron de este tipo, conocidas como *Two Membered Evolution Estrategies* o *(1+1) Evolution Estrategies, (1+1)-ES*, están basadas en un esquema de mutación-selección.

Consiste en una población P^t compuesta por un único individuo a_1^t, los individuos están formados por dos vectores $a_1^t = (x^t, \sigma^t) \ : \ x^t, \sigma^t \in \mathbb{R}^n$. El vector x_1^t es la codificación de una solución candidata (el cromosoma); σ_1^t, el vector de varianzas indica lo lejos que la solución codificada se encuentra de la solución óptima.

$P^0 = a_1^0$ es la población inicial, que consiste en un único individuo que produce un descendiente por medio de mutación, generándose una población intermedia P'^0 de dos individuos. La expresión 2.10 muestra cómo se produce la **mutación** de forma general.

$$\begin{aligned} P^t &= \{a_1^t\}, \quad a_1^t = (x^t, \sigma^t) \\ P'^t &= \{a_1'^t, a_2'^t\} \\ a_1'^t &= a_1^t \\ a_2'^t &= (x'^t, \sigma^t) \end{aligned} \tag{2.10}$$

La mutación se aplica de forma que el descendiente "se parezca" al progenitor y que las variaciones pequeñas sean más frecuentes que los grandes. Al vector x^t se le aplican variaciones aleatorias obtenidas mediante una distribución normal centrada en cero y con

los valores del vector σ^t como desviaciones típicas, 2.11.

$$x^{'t} = x^t + N_0(\sigma^t) \tag{2.11}$$

La **selección** sirve para decidir cuál de los individuos será el que se reproduzca en la siguiente generación, simplemente se elige el individuo cuyo valor de adecuación sea mejor. Siendo la función $f(x)$ la que determine el valor de adecuación, y teniendo en cuenta la solución x tendrá un mejor valor de adecuación cuanto menor sea el valor de $f(x)$, en 2.12 se puede observar cómo se aplica la selección.

$$\begin{aligned} P^{t+1} &= \{a_1^{t+1}\} \\ a_1^{t+1} &= \begin{cases} a_1^{'t} & \text{si } f(x^t) < f(x^{'t}) \\ a_2^{'t} & \text{si } f(x^t) \geq f(x^{'t}) \end{cases} \end{aligned} \tag{2.12}$$

Este proceso de mutación y selección termina cuando se alcanza el criterio de convergencia. Generalmente este criterio consiste en alcanzar un número máximo de generaciones, o que los valores del vector de varianzas sean suficientemente pequeños (nótese que para valores muy pequeños de las varianzas, las variaciones introducidas en la mutación van a ser mínimas).

Por otro lado, el vector de varianzas no debe permanecer fijo, debe modificarse para guiar la búsqueda de la solución óptima. Lo ideal es que produzcan variaciones grandes cuando nos encontramos lejos de la solución óptima y variaciones pequeñas cuando estamos cerca. Para modificar los valores de las varianzas se puede seguir **la regla de 1/5**. Se lleva la cuenta de las veces que ha mejorado la solución candidata, es decir, el número de veces que la población ha cambiado. Si la población ha cambiado m veces en las últimas s generaciones, entonces: $p_s^t = m/s$. El vector de varianzas se actualiza cada s generaciones según la expresión 2.13, donde c_d y c_i son coeficientes de incremento y decremento, respectivamente ($c_d > 1$ y $c_i < 1$).

$$\sigma^{t+1} = \begin{cases} C_d \cdot \sigma^t & \text{si } p_s^t > \frac{1}{5} \\ C_i \cdot \sigma^t & \text{si } p_s^t < \frac{1}{5} \\ \sigma^t & \text{si } p_s^t = \frac{1}{5} \end{cases} \tag{2.13}$$

2.4.2. Estrategias evolutivas múltiples

En los métodos (1+1)-ES, realmente no se utiliza el concepto de población. Con el fin de introducir este concepto, Rechenberg propuso las estrategias evolutivas múltiples, en inglés *multimembered ES*, que más adelante sería denotado por Schwefel como $(\mu+1)$-ES [Schwefel, 1975]. En este método, al introducir la población se posibilitaba introducir una simulación de la reproducción sexual, que fue introducida mediante el operador de cruce. Si $a' = (x', \sigma')$ es el descendiente y $a_a = (x_a, \sigma_a)$ y $a_b = (x_b, \sigma_b)$ son los progenitores, el operador de cruce actúa según la expresión 2.14. Hay que tener en cuenta que X es una variable que toma valores aleatorios en $[0, 1]$.

$$
\begin{aligned}
a' &= \text{cruce}(a_a, a_b) = (x', \sigma') \\
x'_i &= \begin{cases} x_{a,i} & \text{si } X \leq \frac{1}{2} \\ x_{b,i} & \text{si } X > \frac{1}{2} \end{cases} \quad \forall i \in [1, \ldots, n] \\
\sigma'_i &= \begin{cases} \sigma_{a,i} & \text{si } X \leq \frac{1}{2} \\ \sigma_{b,i} & \text{si } X > \frac{1}{2} \end{cases} \quad \forall i \in [1, \ldots, n]
\end{aligned}
\tag{2.14}
$$

Se puede considerar un tipo de cruce uniforme. Por otro lado, ligados a los conceptos de población y reproducción introducidos, los principios de la selección natural inspiraron la selección de los individuos que continuarían en la población. Tras la reproducción, se elimina al individuo con peor valor de adecuación, ya sea el descendiente o uno de los progenitores.

Schwelel extendió el método de $(\mu + 1)$-ES a $(\mu + \lambda)$-ES y (μ, λ)-ES. Esto se hizo para poder explotar los recursos computacionales de los ordenadores paralelos (que en aquel entonces no existían más que de forma teórica) y dotar a las estrategias evolutivas de la capacidad de autoadaptación (del inglés *self-adaptation*). Si se considera σ^t como parte de la información genética de los individuos, debe formar parte de los procesos de mutación y cruce. Asumiendo que los mejores individuos tendrían un valor más adecuado para este parámetro, se esperaba que esos individuos fueran dominando la población, de modo que el ajuste de los valores de sigma emergería por medio de las capacidades autoadaptativas de las que se está dotando a la estrategia evolutiva.

Las estrategias evolutivas del tipo $(\mu + \lambda)$ consisten que una población de μ individuos

produce λ descendientes mediante cruce, a continuación la población de $\mu + \lambda$ individuos se reduce a μ eliminando a los λ individuos con peor valor de adecuación. Esto puede provocar que individuos con valores de adecuación buenos perduren a lo largo de muchas generaciones, lo que puede implicar que se caiga en mínimos locales.

Para evitar este efecto, Schwelel propuso las (μ, λ)-ES. En este caso, λ debe ser mayor que μ necesariamente, pues la vida de los individuos está limitada a una única generación. Todos los progenitores son reemplazados por los descendientes y se seleccionan los μ mejores entre éstos, para que se reproduzcan en la próxima generación.

Se proponen otros tipos de cruce para estos métodos. En la expresión 2.14 se proponía un tipo de cruce equivalente al cruce uniforme que se utiliza cuando, típicamente en algoritmos genéticos se trabaja con cadenas de bits. Pero al utilizar valores reales, es posible combinar los genes de los progenitores para formar nuevos alelos, en lugar de elegir entre dichos genes. Siguiendo la fórmula 2.15 se puede aplicar este tipo de cruce.

$$\begin{aligned} a' &= \text{cruce}(a_a, a_b) = (x', \sigma') \\ x' &= \tfrac{1}{2}(x_a + x_b) \\ \sigma' &= (\sigma_a + \sigma_b)^{\frac{1}{2}} \end{aligned} \tag{2.15}$$

En este caso, la varianza ya no se modifica mediante la regla de $1/5$, también se le aplica mutación. Mientras que los vectores de codificación x^t se mutan de la misma forma, la varianza se muta según la ecuación 2.16, siendo τ la tasa de aprendizaje autoadaptativo.

$$\sigma' = \sigma \cdot e^{N_0(\tau)} \tag{2.16}$$

2.4.3. CMA-ES

El algoritmo CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*), fue propuesto por Hansen y Ostermeir en 1996 [Hansen, 1996]. En esta estrategia evolutiva, se utiliza una matriz de covarianzas, que se va calculando y actualizando durante el proceso para guiar la búsqueda. A continuación se va a explicar la versión del algoritmo que Hansen

y Ostermeir presentarían como extensión de $(1, \lambda)$-ES, aunque en versiones posteriores [Hansen, 2001] se ha presentado el CMA-ES como extensión al método (μ, λ)-ES; pues el concepto subyacente del algoritmo se comprende mejor en esta primera aproximación.

La idea innovadora introducida con esta técnica es que se tiene en cuenta el camino que la población de individuos lleva durante un número determinado de generaciones para ajustar la mutación, mejorando así las capacidades autoadaptativas de la estrategia evolutiva. Se obtiene información del camino que sigue la evolución prestando atención a la correlación. Si generaciones sucesivas presentan correlación paralela (es decir, su producto escalar es mayor que cero) quiere decir que los individuos de esas generaciones están yendo en la misma dirección, por lo que la misma distancia se podría cubrir en menos pasos pero más largos (el camino que sigue la distribución es más largo de lo que debería), mientras que la correlación antiparalela significa que unos pasos anulan a otros, lo que del mismo modo, es ineficiente.

Lo deseable es que no existan correlaciones. Para ello, en este método se sigue el siguiente principio: "La adaptación de la mutación debe reducir la diferencia entre las distribuciones del camino que lleva la evolución y las que llevaría un camino evolutivo bajo selección aleatoria". Esto reduce la correlación entre las mutaciones, pues no están correlacionados bajo selección aleatoria.

Seguir este principio significa que si δ^2 determina la varianza global de la distribución y aparece correlación paralela, se aumenta δ. Si, por el contrario se da correlación antiparalela δ se disminuye. El principio también puede seguirse teniendo en cuenta la forma de la distribución normal, si el camino que sigue la distribución es más largo de lo que debería, la normal se alarga en esa dirección; si por el contrario, es más corto, la normal se aplana. Entonces, la varianza δ y la normal N_0 son dos parámetros que se pueden ajustar para disminuir la correlación entre distintas generaciones.

La distribución normal centrada en 0, puede denotarse en forma matricial $N(0, I)$, donde I es la matriz identidad. Si z es una variable aleatoria que sigue esta distribución, entonces Bz corresponde a la distribución $N(0, BB^T)$. Como se muestra en 2.17, para ajustar la mutación, se utilizan los valores de la varianza global y de la covarianza (BB^T es la matriz de covarianza, y δ la varianza global).

$$x^{t+1} = x^t + \delta^t \, B^t \, z \tag{2.17}$$

donde:

$x \in \mathbb{R}^n$ es el vector a ser optimizado.

$t+1$ denota al *descendiente*.

t denota al *progenitor*.

z es una variable aleatoria con distribución normal.

$\delta > 0$ es la raíz cuadrada de la varianza global.

$B \in \mathbb{R}^{n \times n}$ es una matriz que transforma linealmente a z.

La capacidad autoadaptativa de la estrategia evolutiva reside en aplicar mutación a los parámetros propios de cada individuo o generación de individuos que participan en las operaciones genéticas. En este caso, es preciso mutar la matriz de covarianza y el valor de la varianza global. Estos procesos deben separarse, pues mientras que el valor de σ puede actualizarse en cualquier momento durante la evolución, para actualizar la matriz de covarianza es necesario recoger información sobre la evolución (deben recogerse datos durante al menos $n(n+1)/2$ generaciones) para que se realicen cambios significativos [Hansen, 1996].

El valor de la matriz de covarianza $C = BB^T$, a través de la cual se determina la matriz B, se muta según la expresión 2.18. Para ello se utiliza $s \in \mathbb{R}^n$, que es un vector que almacena una suma ponderada de la mutación aplicada en generaciones anteriores, representa el camino que sigue la evolución; inicialmente $s = 0$ y $C = I$.

$$\begin{aligned} s^{t+1} &= (1-c) \cdot s^t + c_u B^t z \\ C^{t+1} &= (1-c_{\text{cov}}) \cdot C^t + c_{cov} s^t s^{t^T} \end{aligned} \tag{2.18}$$

donde:

$c \in (0, 1]$ es una medida del número de generaciones que se lleva acumulando s.

$c_u = \sqrt{c \cdot (2-c)}$, normalización de la varianza de s resolviendo $1 = (1-c)^2 + c_u^2$.

$c_{cov} \in (0, 1)$ es una medida del número de generaciones transcurridas desde la última actualización de C.

Para determinar B a partir de $C = BB^T$ es necesario resolver una ecuación sin solución única, para ello se emplean los autovalores de C ordenados según sus autovectores.

La varianza global se actualiza de forma similar, se introduce s_δ, de forma análoga a s para la matriz de covarianza y se actualiza el valor de δ siguiendo la expresión 2.19.

$$\begin{aligned} s_\delta^{t+1} &= (1-c)\cdot s_\delta^t + c_u \cdot B_\delta^t z \\ \delta^{t+1} &= \delta^t \cdot \exp\{\beta \cdot (\|s_\delta^t\| - \hat{X}_n)\} \end{aligned} \tag{2.19}$$

donde:

B_δ se corresponde con B con las columnas normalizadas.

$c \in (0, 1]$ es una medida del número de generaciones que se lleva acumulando s_δ.

$c_u = \sqrt{c \cdot (2-c)}$.

β es un parámetro que determina la tasa de cambio de δ.

$\|s_\delta\|$ es la varianza de s_δ.

$\hat{X}_n = \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$

Para realizar las actualizaciones de la matriz de covarianza y la varianza global, se han introducido una serie de parámetros para los que existen valores recomendados que surgieron tras distintas experimentaciones [Hansen, 1996]. El valor de $\hat{X}_n$, $\beta = 1/n$, $c = 1/\sqrt{n}$, y $c_{cov} = 2/n^2$.

Capítulo 3

Objetivos

El objetivo de este proyecto es proponer un método, basado en técnicas de computación evolutiva, para encontrar una medida de distancia que mejore los resultados de clasificación, respecto a los obtenidos mediante un método de clasificación basado en K-Medias utilizando la distancia Euclídea. Para alcanzar esta meta se han fijado los siguientes objetivos:

1. Definir un método de clasificación basado en K-Medias en el que, *a priori*, la optimización de la función de distancia pueda mejorar los resultados obtenidos utilizando la distancia Euclídea.

2. Plantear la optimización de la función de distancia como un problema a resolver con técnicas de computación evolutiva. Es necesario establecer qué técnicas se van a utilizar y definir una representación del problema adecuada a dichas técnicas.

3. Integrar la optimización de la función de distancia en el algoritmo de clasificación propuesto y establecer un método para determinar el funcionamiento y el rendimiento del método frente a distintos dominios de datos.

4. Desarrollar una aplicación que implemente el método propuesto y permita especificar distintas configuraciones (en cuanto a los posibles valores de los parámetros del algoritmo). Es necesario tener en cuenta que el algoritmo propuesto estará sujeto a cambios que van a depender de los resultados obtenidos, por lo que es importante que la aplicación sea fácilmente extensible.

5. Encontrar o diseñar una serie de dominios de datos que, por sus propiedades, permitan comprobar el funcionamiento del algoritmo planteado.

6. Experimentar con al algoritmo propuesto y los dominios encontrados, analizando los resultados con el objetivo de evaluar el método propuesto.

7. A partir del análisis de los resultados, comprobar si se observan las propiedades esperadas en el algoritmo y plantear posibles alternativas para solventar las carencias que presente.

Capítulo 4

Desarrollo

En este capítulo se explica el proceso de diseño e implementación del método propuesto. Para ello se comienza describiendo en qué consiste el algoritmo y qué técnicas de computación evolutiva se emplean. A continuación, se explica el proceso de desarrollo de la aplicación que implementa dicho algoritmo. Una vez definido el método, se proponen una serie de dominios con los que experimentar, comentando sus propiedades y las razones por las que se ha decidido utilizarlos.

Posteriormente, se especifican y analizan los resultados obtenidos con los dominios propuestos. El análisis de los resultados permitirá confirmar ciertas propiedades esperadas en el algoritmo, así como determinar ciertas carencias para las que se propondrán alternativas.

4.1. Descripción del método propuesto

El algoritmo de clasificación propuesto es un método de aprendizaje supervisado basado en K-Medias, en el que se incluye la optimización de la función de la función de distancia mediante el uso de técnicas de computación evolutiva.

En primer lugar, se va a explicar en qué consiste la técnica de clasificación, para, a continuación, describir cómo se resuelve la optimización de la función de distancia mediante el uso de estrategias evolutivas.

4.1.1. El método de clasificación

De forma básica, el algoritmo en consiste encontrar un serie de puntos representativos de los datos que ya tenemos clasificados (a los que llamaremos centros), para poder establecer la clase de nuevos datos en base a la proximidad de estos puntos.

Si contamos con un conjunto de datos que pertenecen a n clases, al que denominamos X, y definimos el subconjunto de elementos que pertenecen a la clase i como $X_i \in X \ : \ i \in \{1, 2, ..., n\}$. La forma de calcular los centros va a consistir en ejecutar el método K-Medias con k prototipos tantas veces como clases haya.

De esto resultan k centros representativos de cada clase, en total $n \cdot k$ centros para todo el conjunto de datos. Cuando se pretenda clasificar una nueva muestra, se considerará que pertenece a la misma clase que le correspondía al centro más cercano. Es decir, si $c_{i,j}$ (el centro número j de la clase i) es el más cercano al nuevo dato y, se determina que y pertenece a la clase i.

Para determinar la proximidad entre los datos y los centros se empleará la función de Distancia Euclídea Generalizada optimizada.

En la figura 4.1 se muestra visualmente el funcionamiento del método de clasificación. Si se establece $k = 2$, entonces, al haber dos clases en el conjunto de datos, el algoritmo K-Medias se ejecuta dos veces. Una vez calculados todos los centros podemos clasificar nuevos datos. Para clasificar el punto representado de color verde, se calcula la distancia desde dicho punto a todos los centros utilizando la función de distancia que corresponda. En este caso se ha utilizado la Distancia Euclídea, resultando que se le asigna la clase azul, por ser la clase correspondiente al centro más cercano.

Se ha elegido K-Medias como método para calcular los centros por tratarse de una técnica de *clustering* muy utilizada por obtener buenos resultados y converger rápidamente. Al utilizar una técnica de agrupamiento para calcular los centros, se trata de descubrir, en cada clase, las agrupaciones de datos más próximos entre sí (según nuestra métrica de distancia) y utilizar como centro el punto que mejor represente estos grupos de datos.

El número de centros por clase, es decir el parámetro k, va a depender del dominio que se utilice, por lo que tendrá que determinarse de forma experimental. Hay que tener en cuenta que si $k = 1$, los centros corresponden a la media aritmética de los datos pertenecientes a

cada subconjunto X_i, dicho de otro modo, corresponden a la media de los datos de cada clase. Por otro lado, según se va aumentando el valor de k, va aumentando la complejidad del algoritmo; no sólo en lo que respecta a la ejecución de K-Medias para calcular los centros, sino también a la hora de optimizar la función de distancia.

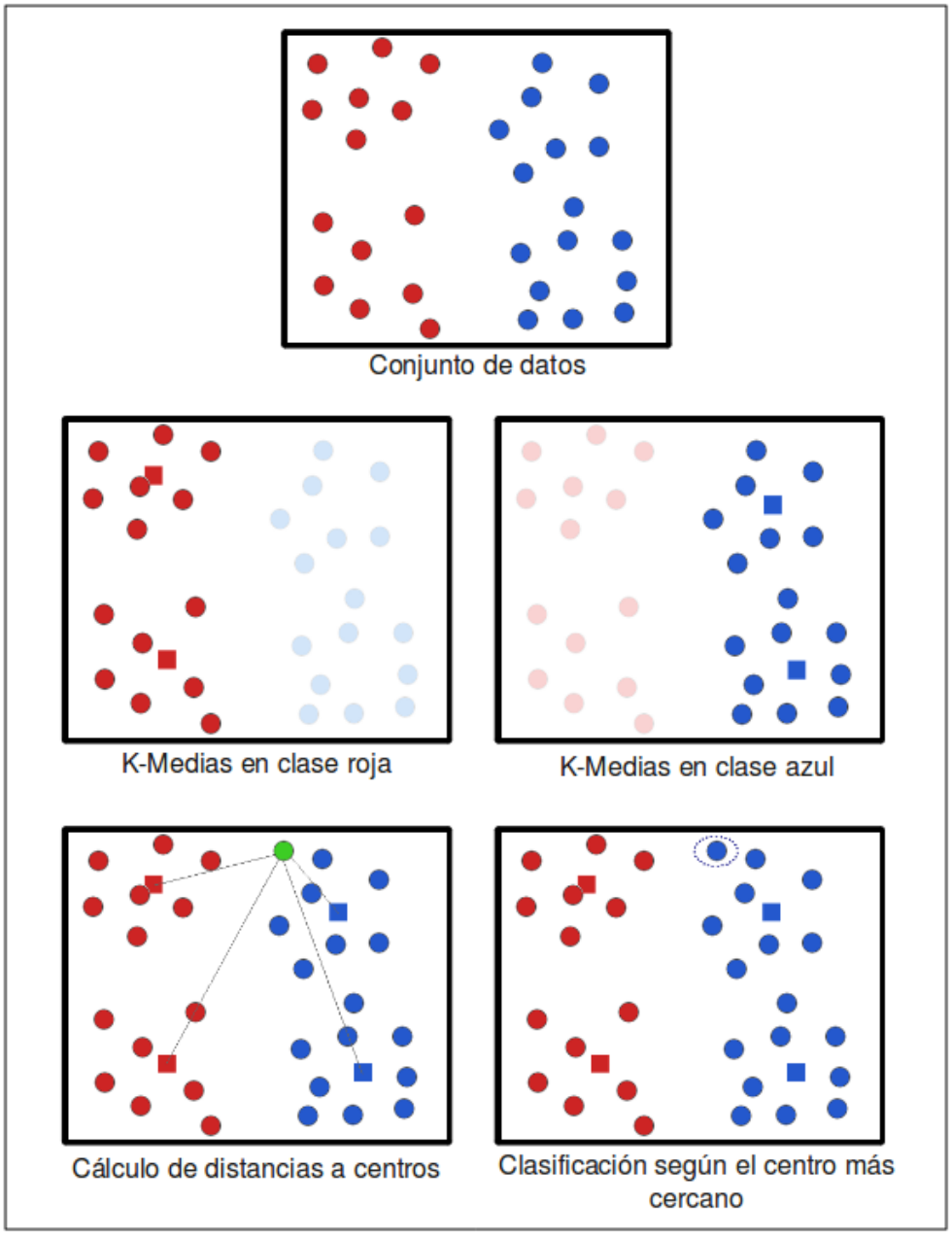


Figura 4.1: Ejemplo de funcionamiento del método de clasificación.

4.1.2. Optimización de la función de distancia

Como métrica de distancia se va a utilizar la Distancia Euclídea Generalizada, que está caracterizada por la matriz simétrica $M = M_d \cdot M_d^T \in \mathbb{R}^{n \times n}$.

Resolver el problema de la optimización de la función de distancia para un determinado dominio, consiste en encontrar la matriz que defina la GED de forma que el número de aciertos en la clasificación sea máximo.

Los valores de los que consta la matriz son reales. Entre las técnicas de computación evolutiva, las estrategias evolutivas fueron definidas, precisamente, para optimizar vectores de valores reales; por lo que la búsqueda de la matriz óptima se va tratar de resolver mediante este tipo de técnicas. Concretamente, se van a emplear los siguientes tipos de estrategias evolutivas:

- **CMA-ES**: se trata de una estrategia evolutiva muy utilizada en los últimos años ya que sus capacidades autoadaptativas hacen que pueda obtener buenos resultados en tareas de optimización. Además ya ha sido utilizada en problemas de transformaciones de datos para tareas de clasificación [Valls, 2009].
- **EE-(1+1)** (*(1+1)-ES*, por sus siglas en inglés): es la técnica más sencilla entre las estrategias evolutivas. Resulta fácil de implementar y no tiene demasiados parámetros que configurar, por lo que se considera apropiado experimentar con este tipo de técnicas.
- **Estrategias evolutivas múltiples**: introducen los conceptos de población y reproducción, y permiten dotar a las estrategias evolutivas de ciertas capacidades de autoadaptación. Puede ser posible, por lo tanto, obtener buenos resultados en casos en los que las técnicas EE-(1+1) no funcionen.

Pese a las peculiaridades de cada una de las estrategias evolutivas, la representación de la matriz de la GED a ser optimizada es común. A la hora de elegir esta representación, es necesario tener en cuenta todas las características del problema.

En primer lugar, es lo mismo caracterizar la Distancia Euclídea Generalizada por la matriz M o por la matriz M_d, ya que $M = M_d \cdot M_d^T$. Como la matriz M es simétrica, los elementos de la matriz triangular superior correspondiente son los mismos que los de la

matriz triangular inferior, por lo que si se utiliza esta matriz, sólo es necesario optimizar $\sum_{i=0}^{n} i$ valores, mientras que si se utilizase la matriz M_d habría que optimizar n^2 valores.

Por otro lado, si la matriz M es diagonal, la GED corresponde a la Distancia Euclídea Ponderada. En algunos dominios, es posible que la Distancia Euclídea Ponderada sea suficiente para obtener buenos resultados. En este caso, el número de valores a optimizar se reduce a n, lo que facilita en gran medida la búsqueda de la matriz óptima. Por lo tanto, se van a considerar dos alternativas: utilizar matrices diagonales o utilizar matrices simétricas.

Otro punto a tener en cuenta es el número de funciones de distancia a utilizar. Es necesario calcular la distancia entre los puntos y los centros, por lo que se podrían utilizar tantas funciones de distancia como centros. Teniendo en cuenta que va a haber k centros por clase, habría que encontrar $c \cdot k$ matrices de distancia. En este caso, también se van a considerar dos posibilidades: función de distancia por cada centro ($c \cdot k$ matrices), o utilizar una función de distancia por clase (c matrices).

Tras haber analizado las características del problema, se puede definir el cromosoma que se va a evolucionar mediante las estrategias evolutivas. Denotamos al cromosoma como el vector C, que va a consistir en la consecución de los valores necesarios para representar o bien c matrices o bien $c \cdot k$ matrices.

Definimos c_i, como el fragmento del cromosoma C correspondiente a la matriz M_i, correspondiéndose el cromosoma con la expresión 4.1.

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_m \end{bmatrix} \tag{4.1}$$

donde:

$$m = \begin{cases} c \text{ , si se utiliza una matriz por clase.} \\ \\ c \cdot k \text{ , si se utiliza una matriz por centro.} \end{cases}$$

Si definimos la matriz M_i de la forma siguiente:

$$M_i = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n} \end{bmatrix}$$

La correspondencia entre c_i y M_i se establece según la expresión 4.2 si se utilizan matrices diagonales y 4.3 si se utilizan matrices simétricas (en este caso se utilizan los valores de la matriz triangular superior correspondiente).

$$c_i = \begin{bmatrix} m_{1,1} & m_{2,2} & \cdots & m_{n,n} \end{bmatrix} \tag{4.2}$$

$$c_i = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} & \cdots & m_{j,j} & \cdots & m_{j,n} \cdots & m_{n,n} \end{bmatrix} \tag{4.3}$$

Además de la representación de las matrices de distancia para su optimización, otro aspecto a tener en cuenta son las restricciones que van ligadas a las matrices de distancias. Para que la función de distancia se considere como tal, la matriz que la define debe ser necesariamente definida positiva.

El hecho de satisfacer esta restricción implica mucho tiempo extra de cómputo a la hora de evaluar los individuos, por lo que esta restricción va a relajarse tal como se indica a continuación: todas las matrices con valores reales serán admitidas. Esto implica que es posible que el cuadrado de la distancia para ciertos puntos sea negativo. En estos casos se considerará que dichos puntos están infinitamente alejados. Esta restricción se resume en la ecuación 4.4.

$$d(x,y)_M = \begin{cases} \sqrt{(x-y)\ M\ (x-y)^T}\ \text{, si } (x-y)\ M\ (x-y)^T \ \geq\ 0 \\ \\ \infty\ \text{, en otro caso.} \end{cases} \tag{4.4}$$

Esta forma de calcular las distancias, relajando algunas de sus propiedades, puede mejorar los resultados de clasificación para ciertos tipos de dominios. Pero, para los casos en los que sea necesario encontrar una medida de distancia que cumpla todas las propiedades

de las funciones de distancias, será posible restringir que todos los valores de las matrices sean mayores o iguales a cero.

La implementación de esta restricción es sencilla y no requiere hacer costosos cálculos durante el proceso de evolución. Consiste en utilizar el valor absoluto de cada elemento del cromosoma a la hora de obtener las matrices correspondientes.

Por otro lado, utilizar sólo matrices cuyos elementos son no negativos implica acotar el espacio de búsqueda, ya que hay matrices definidas positivas que tienen elementos menores que cero. Además, el hecho de utilizar los valores absolutos de cada valor introduce redundancias en la codificación, de forma que el espacio de búsqueda aumenta innecesariamente.

4.1.3. Evaluación del aprendizaje

Tras definir el algoritmo de clasificación es preciso establecer un método de evaluación que permita medir el funcionamiento del aprendizaje automático frente a distintos dominios.

En primer lugar, consideraremos los resultados obtenidos utilizando la Distancia Euclídea frente a la Distancia Euclídea Generalizada optimizada. El objetivo será mejorar los resultados que se obtengan mediante la Distancia Euclídea; en otro caso, la optimización de la matriz de distancias carecería de sentido.

Por otro lado, es necesario definir cómo se van a utilizar los datos disponibles. Para poder evaluar el aprendizaje, es necesario contar con una serie de datos de los que extraer conocimiento y otros datos que clasificar en base al conocimiento extraído. Dicho de otro modo, es preciso contar con un conjunto de entrenamiento y un conjunto de validación. Para evitar que, al dividir los datos en estos dos conjuntos, aparezcan sesgos, se va a utilizar el método de la validación cruzada.

La validación cruzada consiste en dividir los datos en n subconjuntos, de forma que se ejecutan el entrenamiento y la validación n veces. En cada ejecución se utiliza un subconjunto para realizar la validación tras haber efectuado el aprendizaje con el resto de los datos. Finalmente, se construye el clasificador utilizando todos los datos y se estima el porcentaje de aciertos como la media del obtenido en todas las validaciones realizadas.

Por último, es importante señalar, que aún utilizando el método de validación cruzada, podrían aparecer problemas causados a la hora de realizar la evaluación si los datos están ordenados. Para evitar este tipo de problemas, los datos se distribuirán de forma aleatoria.

4.2. Definición de la aplicación

En esta sección se explican las características generales del sistema desarrollado. Esto comprende las funcionalidades que ofrece la aplicación, así como la plataforma y herramientas de desarrollo utilizadas para realizar la implementación.

4.2.1. Capacidades y restricciones generales

Con el desarrollo de esta aplicación se pretende implementar el algoritmo de clasificación objeto de este proyecto, permitiendo realizar una experimentación para poder evaluarlo. A su vez, esta aplicación debe ser extensible y modificable. Así, en base a los resultados obtenidos, posibles cambios que sean definidos en el método podrán hacerse efectivos en la aplicación de una forma sencilla.

Con este objetivo, la aplicación debe mostrar las siguientes funcionalidades:

1. Debe implementar el algoritmo descrito en la sección 4.1 *Descripción del método propuesto*, incluyendo todas las características y técnicas especificadas.
2. Permitirá especificar los parámetros de configuración del algoritmo y de las estrategias evolutivas, de las cuales se haga uso, de una forma sencilla y flexible. Concretamente, se permitirá especificar los siguientes parámetros:
    - Si se utilizan matrices de distancia simétricas o diagonales.
    - Si se va a utilizar una matriz por cada clase o por cada centro.
    - El número de subconjuntos a emplear en la validación cruzada.
    - El criterio de convergencia para el algoritmo K-Medias.
    - El cromosoma inicial, la varianza inicial y el número máximo de iteraciones para el algoritmo CMA-ES.
    - El cromosoma inicial, la varianza inicial, el criterio de convergencia y el número máximo de iteraciones para el algoritmo EE-(1+1).
    - El cromosoma inicial, la varianza inicial, el criterio de convergencia, el número máximo de iteraciones y el método de selección para las estrategias evolutivas múltiples.

3. Los valores de los parámetros se especificarán mediante un fichero de configuración.

4. Será posible comparar los resultados obtenidos mediante el método de clasificación en el que se utiliza la Distancia Euclídea y los obtenidos al utilizar la GED optimizada utilizando estrategias evolutivas.

5. Se podrán comparar los resultados de clasificación utilizando la GED optimizada mediante las distintas estrategias evolutivas propuestas, o mediante distintas configuraciones de la misma estrategia evolutiva.

6. Debe mostrar el tiempo de ejecución del algoritmo, excluyendo el tiempo empleado para realizar la clasificación con la Distancia Euclídea.

7. Será posible obtener la matriz de la Distancia Euclídea Generalizada optimizada mediante las estrategias evolutivas, así como los centros calculados mediante el algoritmo K-Medias, que permiten realizar la clasificación.

8. No presentará limitaciones en cuanto al número de datos utilizados o el número de atributos de los mismos. Estas limitaciones vendrán dadas por la memoria del sistema en el que se ejecute la aplicación.

4.2.2. Plataforma y herramientas de desarrollo

Para el desarrollo de la aplicación se emplea el lenguaje de programación Java. Este lenguaje de programación fue desarrollado por Sun Microsystems a principios de los años 90. Fue definido dentro del paradigma de la orientación a objetos y con el objetivo de permitir ejecutar un mismo programa en diferentes sistemas operativos [Gosling, 1996]. Para ello, el código se compila en *bytecodes* que son interpretados por la Máquina Virtual Java (*JVM*, Java Virtual Machine) [Lindholm, 1999].

Este lenguaje, además de aportar las ventajas de la orientación a objetos y permitir desarrollar aplicaciones bastante portables, proporciona una extensa interfaz de programación de aplicaciones (del inglés *API, Application Programming Interface*). A su vez, al tratarse de un lenguaje muy utilizado, es posible encontrar un gran número de utilidades y bibliotecas, desarrolladas en este lenguaje para ámbitos diversos.

Además del API de Java[1], se han utilizado las siguientes bibliotecas y programas desarrollados por terceros:

[1] `http://java.sun.com/javase/6/docs/api/`.

JAMA: A Java Matrix Package[2]**.** Se trata de una biblioteca Java que proporciona un conjunto de utilidades sobre álgebra lineal. Se utiliza esta implementación para efectuar las operaciones de álgebra matricial.

CMA Evolution Strategy Source Code[3]**.** Es una implementación en Java del algoritmo CMA-ES, proporcionada por uno de los autores de dicho algoritmo.

Normal Distribution Random Number Generator[4]**.** Implementación Java de un generador de números aleatorios que siguen una distribución Normal.

Para la implementación de la aplicación se ha utilizado el entorno de desarrollo integrado (*IDE, Integrated Development Environment*) de código abierto Eclipse[5]. Algunas de las características más destacables de este entorno de programación son las siguientes: dispone de un editor de textos con resaltado de sintaxis, permite control de versiones CVS[6] asistentes y plantillas para la creación de proyectos y clases, compilación "en tiempo real" que permite detectar errores de sintaxis según se escribe el código y dispone de un amplio conjunto de extensiones.

Otras herramientas que se han utilizado para el desarrollo de esta aplicación son las siguientes:

GNU-Octave[7]**.** Lenguaje y plataforma de libre distribución para la resolución de problemas numéricos con compatibilidad con la herramienta Matlab. Se ha utilizado para realizar prototipos y algunos experimentos rápidos.

GNU-Plot[8]**.** Herramienta para la representación de datos y funciones matemáticas. Se ha utilizado para representar gráficamente y visualizar conjuntos de datos en dos y tres dimensiones, además de emplearse indirectamente al estar integrada con GNU-Octave.

[2] `http://math.nist.gov/javanumerics/jama/`.

[3] `http://www.lri.fr/~hansen/cmaes_inmatlab.html`.

[4] Obtenido de la biblioteca PSimJ2 de simulación en Java: `http://science.kennesaw.edu/~jgarrido/psimj.html`.

[5] `http://www.eclipse.org`

[6] *Concurrent Versions System*, sistema informático de control de versiones

[7] `http://www.gnuplot.info/`.

[8] `http://www.gnu.org/software/octave`.

4.3. Diseño de la aplicación

En esta sección se define la estructura del sistema y se especifican los componentes de los que consta, así como las relaciones entre ellos. También se detallan algoritmos y otros detalles relativos a la implementación de la aplicación.

4.3.1. Estructura de la aplicación

La estructura general de la aplicación se establece mediante la descomposición de la misma en subsistemas. Esto hace posible comprender de qué elementos consta la aplicación y cuáles son las relaciones entre ellos. La figura 4.2 es una representación gráfica de esta descomposición en subsistemas.

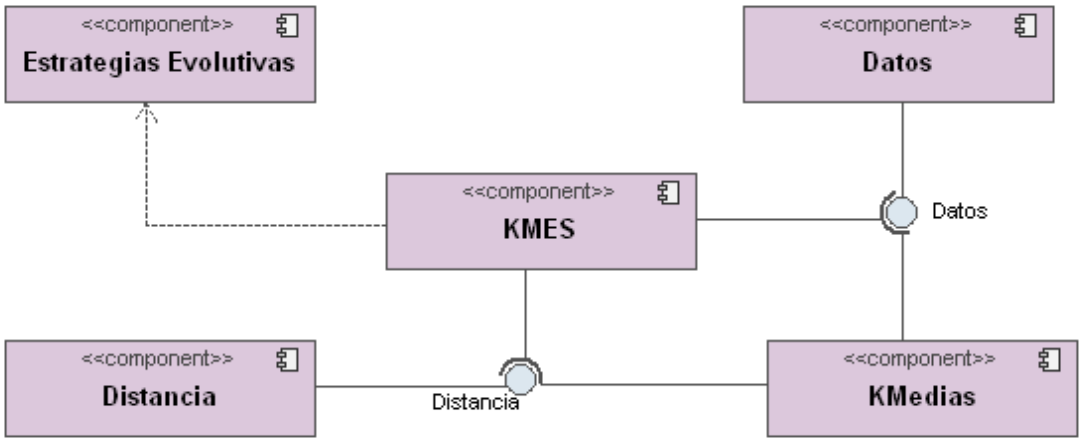


Figura 4.2: Arquitectura de la aplicación.

La estructura lógica del sistema corresponde a los siguientes subsistemas:

1. **Estrategias Evolutivas**: este subsistema corresponde a la implementación y uso de los distintos tipos de estrategias evolutivas empleadas en la aplicación.
2. **Distancia**: corresponde a la implementación de la función de Distancia Euclídea Generalizada, así como a las distintas operaciones a realizar relativas a las funciones de distancias. Ofrece una interfaz bien definida que será utilizada por otros subsistemas.

3. **Datos**: en este subsistema se implementa la carga en memoria de las muestras de datos y las operaciones realizadas con los mismos. Al igual que el subsistema Distancia, ofrece una interfaz bien definida.

4. **KMedias**: la implementación del algoritmo K-Medias es el objeto de este subsistema. Este subsistema está directamente relacionado con los subsistemas Distancia y Datos, pues en él se hace uso de las interfaces que definen.

5. **KMES**: en este subsistema se hace la implementación, propiamente dicha, del método de clasificación propuesto. Para ello se hace uso de elementos del resto de subsistemas (Estrategias Evolutivas, funciones de Distancia, dominios de Datos, y el algoritmo K-Medias), por lo que presenta relaciones y dependencias con los mismos.

4.3.2. El subsistema Estrategias Evolutivas

Este subsistema contiene la implementación de las Estrategias Evolutivas empleadas. Se implementa como un paquete independiente que podría ser utilizado en otras aplicaciones.

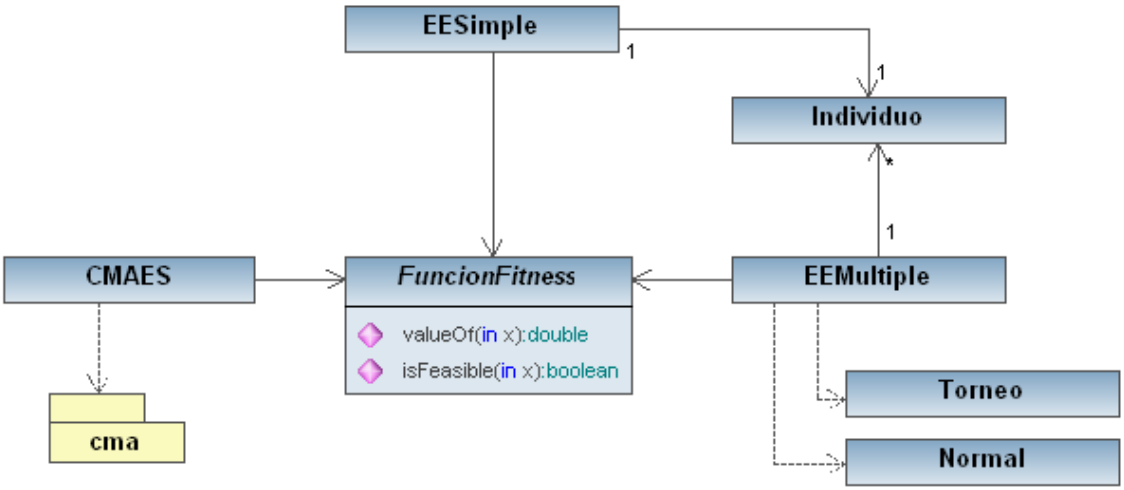


Figura 4.3: Subsistema Estrategias Evolutivas.

El diagrama de la figura 4.3 muestra las clases de las que consta este componente y las relaciones entre ellas.

A continuación, se explica cada uno de los elementos de este subsistema:

FunctionFitness: se trata de una clase abstracta que define las posibles funciones de *fitness*. Si se quieren emplear las estrategias evolutivas incluidas en este subsistema, se debe implementar una clase que represente el valor de adecuación de los individuos, la cual debe heredar de *FunctionFitness*. Únicamente se definen dos funciones:

- valueOf: la implementación de esta función constará de la propia definición de la función de *fitness*, es de decir, es en la que se calculará el valor del individuo (representado como el cromosoma que la función recibe como parámetro).
- isFeasible: esta función permite determinar si un individuo es o no un posible candidato de solución. Recibe el cromosoma que caracteriza al individuo y devuelve *true* en caso de que el individuo sea válido y *false*, en caso contrario.

CMAES: esta clase actúa como *adaptador* que permite utilizar la implementación del algoritmo CMA-ES (CMA Evolution Strategy Source Code) que está incluida en el paquete cma. De esta forma, las dependencias con este elemento externo sólo están presentes en la clase CMAES. Por otro lado, ofrece una interfaz más sencilla de utilizar y emplea la clase correspondiente como representación del fitness.

EESimple: se trata de la implementación de las Estrategias Evolutivas de tipo EE-(1+1). Al igual que el resto de clases que implementan o adaptan este tipo de algoritmos, utiliza la clase FunctionFitness y ofrece una interfaz fácil de emplear.

EEMultiple: la clase EEMultiple implementa las Estrategias Evolutivas Múltiples (de los tipos EE-(μ, λ) y EE-$(\mu + \lambda)$). Siempre se emplean Torneos como estrategias de selección, salvo que se especifique que todos los individuos de la población se reproducen en cada generación. Por otra parte, también hace uso de FunctionFitness y ofrece una interfaz sencilla.

Individuo: la representación de los individuos (representados por un vector correspondiente a su cromosoma y un vector de varianzas) se realiza utilizando esta clase. Además de estos dos elementos, también se almacena el valor de adecuación. Como se aprecia en el diagrama de la figura 4.3, las Estrategias Evolutivas Simples constan de un solo individuo, mientras que en las Estrategias Evolutivas Múltiples hay tantos individuos como indique el tamaño de la población. Implementa la interfaz *Comparable*[9], permitiendo comparar los individuos según su valor de adecuación.

[9]Clase Comparable<E>, definida en el paquete java.lang del *Java Development Kit 6.0*.

Torneo: clase en la que se implementa el Torneo como método de selección de individuos para la reproducción. Esta implementación es empleada en las Estrategias Evolutivas Múltiples.

Normal: esta clase permite obtener valores aleatorios según en una distribución normal. Es empleada para la mutación de las varianzas en las Estrategias Evolutivas Múltiples.

4.3.3. El subsistema Distancia

El objetivo de este subsistema es proporcionar todas las utilidades precisas respecto a las funciones de distancia. Esto se traduce en la implementación de la función de Distancia Euclídea Generalizada, caracterizada por una matriz, que permita obtener el valor de la distancia de dos puntos cualesquiera.

Esta implementación se ha realizado en la clase **GED**, en la cual se hace uso del paquete JAMA para realizar las operaciones con matrices que sean necesarias para calcular la distancia. Las características más importantes de dicha clase son las siguientes:

- Cada instancia de la clase GED define, la función de Distancia Euclídea Generalizada caracterizada por una matriz $M \in \mathbb{R}^{n \times n}$. Por lo tanto, permite calcular la distancia de dos puntos x e $y \in \mathbb{R}^n$:

$$d_M(x, y) \ = \ \sqrt{(x - y)M(x - y)^T}$$

- Proporciona una función para obtener, automáticamente una instancia de GED que represente la Distancia Euclídea (siendo M la matriz identidad), de forma que únicamente es necesario indicar la dimensión de la matriz para obtener la instancia.
- Es posible obtener la ecuación correspondiente a la matriz M_d, donde $M = M_d \cdot M_d^T$, al proporcionarse un método para obtener el producto de una matriz por su traspuesta.

4.3.4. El subsistema Datos

Este subsistema permite a la aplicación interactuar con los patrones de los diferentes dominios de datos. Para ello, se implementa una representación de cada patrón, así como

las distintas operaciones que es necesario realizar con los datos de entrada para poder utilizarlos en el resto del programa.

En la figura 4.4 se muestran las clases utilizadas con este fin. La clase **Patron** se utiliza como la representación de cada muestra de datos y en la clase **Util** se implementan todas las operaciones que es necesario realizar con conjuntos de patrones.

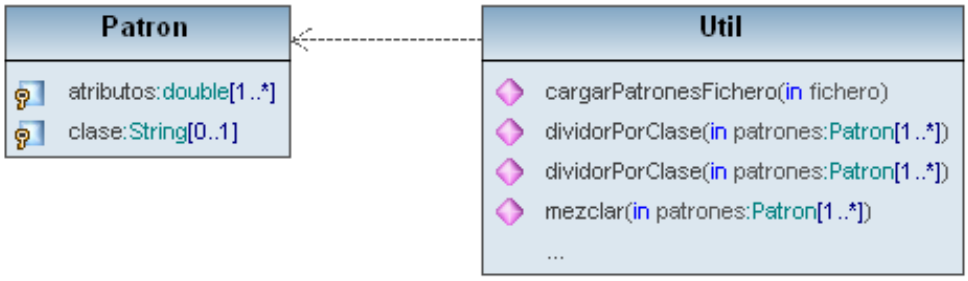


Figura 4.4: Subsistema Datos.

Los patrones de datos vienen definidos por un vector de valores reales y la clase a la que pertenecen. La clase Patron también permite representar datos no clasificados (para los que no se ha definido la clase), pero esta característica ha sido incluída para facilitar la extensibilidad y no va a ser utilizada en este contexto.

La operaciones más importantes definidas en la clase Util son las siguientes:

- Se permite cargar los datos de un dominio desde un fichero. Los atributos de cada patrón de datos estarán separados por espacios y/o tabuladores, y cada patrón estará separado de los demás por un salto de línea (En el anexo A *Manual de usuario* se pueden consultar los detalles del formato de los ficheros de datos).

- Es posible ordenar los datos cargados de forma aleatoria.

- Se pueden dividir los datos en subconjuntos, según la clase a la que pertenezcan.

- Se proporciona una operación para obtener el conjunto de clases correspondiente a un conjunto de datos cargado.

4.3.5. El subsistema KMedias

En el subsistema KMedias se realiza la implementación del algoritmo K-Medias de la que se hace uso en la aplicación. Esta implementación se hace efectiva en la clase **KMedias**, cuyas características principales se especifican a continuación:

- Al crear una instancia de la clase KMedias (para realizar una ejecución del algoritmo), es necesario proporcionar los siguientes elementos: el conjunto de datos, el número de centros a utilizar y el margen de error utilizado en la condición de parada.
- Como función de distancia, se emplea la Distancia Euclídea Generalizada. Por defecto se emplea la matriz identidad, correspondiente a la Distancia Euclídea, pero es posible especificar funciones GED diferentes para calcular la distancia de los datos a cada centroide. Esto presenta el siguiente problema: si alguna de las funciones especificadas no cumple todas las propiedades de las funciones de distancia, no se garantiza la convergencia del algoritmo K-Medias. Para detectar esta situación, se restringe el número de ejecuciones del algoritmo a 10000, de modo que si se alcanza este número de iteraciones se concluye que la ejecución de K-Medias no ha concluido correctamente y se muestra un mensaje de aviso por la salida estándar.
- Antes de comenzar la ejecución del algoritmo, se debe realizar la inicialización de los centros (también es posible proporcionar una serie de puntos como centros, que ya hayan sido calculados previamente). Se incluyen tres posibles métodos de inicialización de los centroides:
  - Inicialización aleatoria 1: se asignan valores aleatorios dentro de un rango definido por los valores máximos y mínimos en cada atributo de los datos como centroides.
  - Inicialización aleatoria 2: a cada centroide se le asigna como valor inicial el correspondiente a un patrón del conjunto de datos.
  - Inicialización K-Medias++[Arthur, 2007]: se sigue el algoritmo descrito en la sección 2.1 *El algoritmo K-Medias* del contexto. A continuación, se define de forma más detallada la inicialización de los centros utilizando esta técnica.

Sea $X = x_1, ..., x_n$ el conjunto de datos para el que definir k centros y sea C el conjunto de centroides ya definidos.

1. Se elige de forma aleatoria $x_i \in X$ como primer centro c_1.

2. Se calcula $D(x)$ para todos los puntos del conjunto X, donde $d(x, y)$ denota la función de distancia.

$$D(x) = \min_{\forall c_j \in C} \{d(x_i, c_j\}$$

3. Se elige un valor aleatorio y en el rango $[0, \sum_{i=0}^{n} D(x_i)^2]$.
4. Se encuentra el punto $x_i \in X$ que cumpla:

$$\sum_{j=0}^{i-1} D(x_j)^2 < y \leq \sum_{j=0}^{i} D(x_j)^2$$

5. Se añade el punto x_i con conjunto C.
6. Se repiten los pasos de 2 a 5 hasta encontrar los k centros.

Es necesario hacer una implementación del algoritmo eficiente para que resulte útil inicializar los centros mediante esta técnica. Para ello se siguen principios de programación dinámica, de forma que se almacenan y reutilizan los valores calculados en iteraciones anteriores. Si definimos

$$S = \{S_1, ..., S_n\} \ : \ S_i = \sum_{j=0}^{i} D(x_j)^2,$$

los valores del subconjunto S se pueden calcular de la forma siguiente:

$$S_1 = D(x_1)^2$$
$$S_i = S_{i-1} + D(x_i)^2 \ \forall \ i > 1$$

Por otra parte, encontrar el índice del valor x_i para el que $S_{i-1} < y \leq S_i$, se reduce a realizar una búsqueda binaria[10]entre los valores del conjunto S.

- Se permiten dos modos de ejecución: el modo normal y el modo verboso (del inglés, *verbose*) en el que se muestran, por la salida estándar, trazas de todos los pasos de ejecución del algoritmo.

4.3.6. El subsistema KMES

Este subsistema contiene la implementación del algoritmo propuesto, para ello se hace uso de las herramientas que el resto de subsistemas ofrecen (manejo de funciones de distancia, estrategias evolutivas y del algoritmo K-Medias).

[10]Se emplea el la función binarySearch de la clase Arrays definida en el paquete java.util del *Java Development Kit 6.0*.

En la figura 4.5 se muestra, de forma general, las partes de las que consta la implementación de la técnica propuesta.

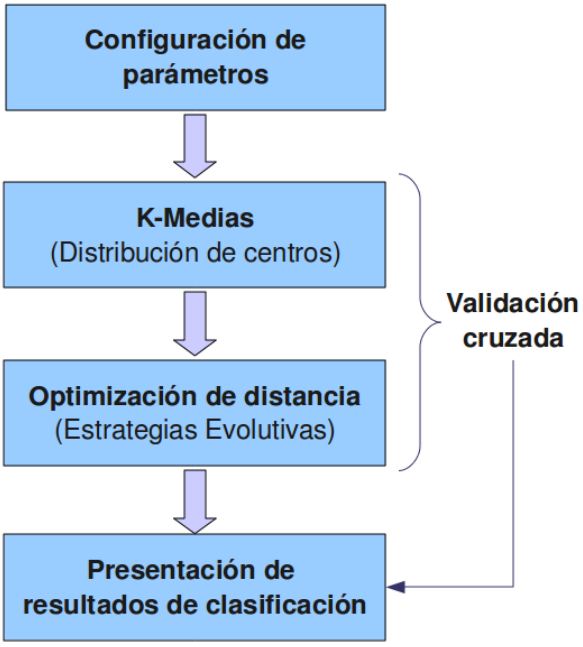


Figura 4.5: Funcionamiento general de la aplicación KMES.

Dentro del subsistema KMES se pueden distinguir dos partes bien diferenciadas: la carga de los parámetros de configuración, la especificación de funciones de fitness.

En la figura 4.6 se muestran las clases relativas a la carga de los parámetros de ejecución, así como las relaciones entre dichas clases.

La clase **Configuracion** permite cargar los parámetros de configuración del programa desde un fichero. Todas las características configurables del método propuesto son parámetros que se pueden cargar. Esto incluye los tipos de estrategias evolutivas a utilizar y su configuración concreta; para representar la configuración de las estrategias evolutivas se emplean las clases derivadas de **Algoritmo**, que se emplea la clase Configuracion.

Para efectuar la carga de la configuración se emplean las *Properties*[11] de Java, lo que facilita la carga de datos cuando se han de especificar variables y valores. En el anexo A. *Manual de usuario* se detalla la estructura del fichero de configuración de la aplicación.

Para implementar la técnica de clasificación descrita, resulta imprescindible definir las funciones de fitness que se emplearán en las estrategias evolutivas empleadas. El diagrama

[11]Clase Properties definida en el paquete java.util del *Java Development Kit 6.0*.

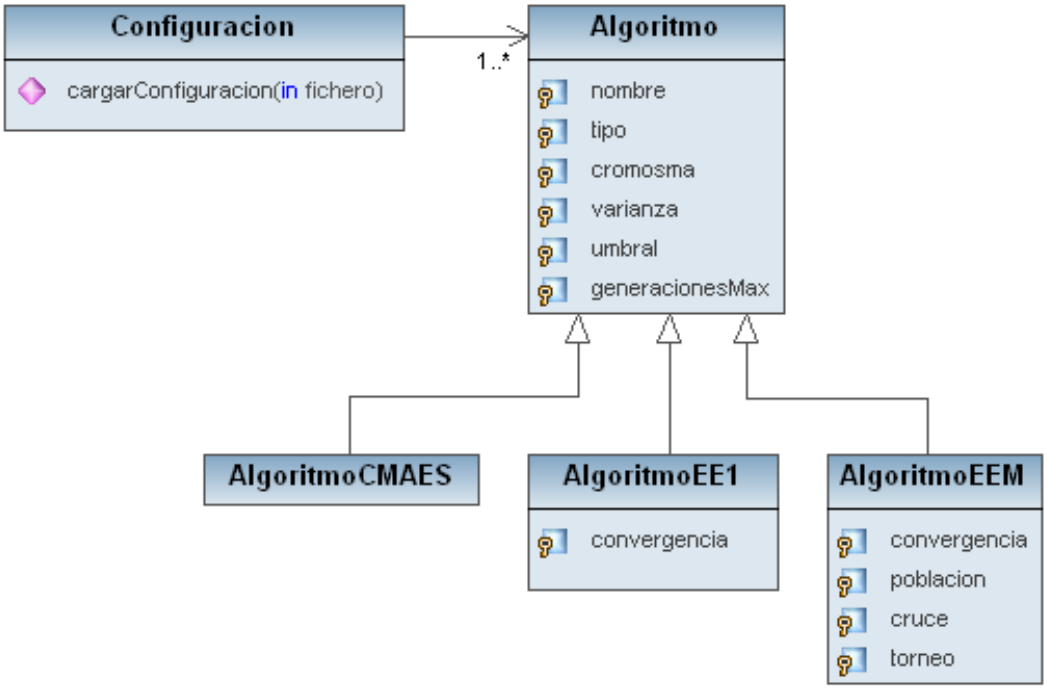


Figura 4.6: Subsistema KMES: carga de parámetros de configuración.

de la figura 4.7 muestra la estructura de clases definida con este fin.

Para emplear el subsistema Estrategias Evolutivas, todas las funciones de fitness empleadas heredan de la clase **FunctionFitness**, perteneciente a dicho subsistema.

La clase **FitnessKmes** incluye todas las operaciones que deberán ser comunes para cualquier función de fitness que se vaya a utilizar en la aplicación:

- Permite especificar y cargar una serie de patrones de datos que se utilizarán para determinar el fitness según la tasa de aciertos del método de clasificación.
- Es posible especificar la dimensión de los datos y las matrices a evolucionar.
- Se puede introducir el número de matrices a evolucionar que contendrá la codificación del cromosoma en las Estrategias Evolutivas, así como el número de centros a evolucionar, si éstos también estuvieran contenidos en el cromosoma.
- Proporciona operaciones para obtener los elementos codificados a partir de un cromosoma.

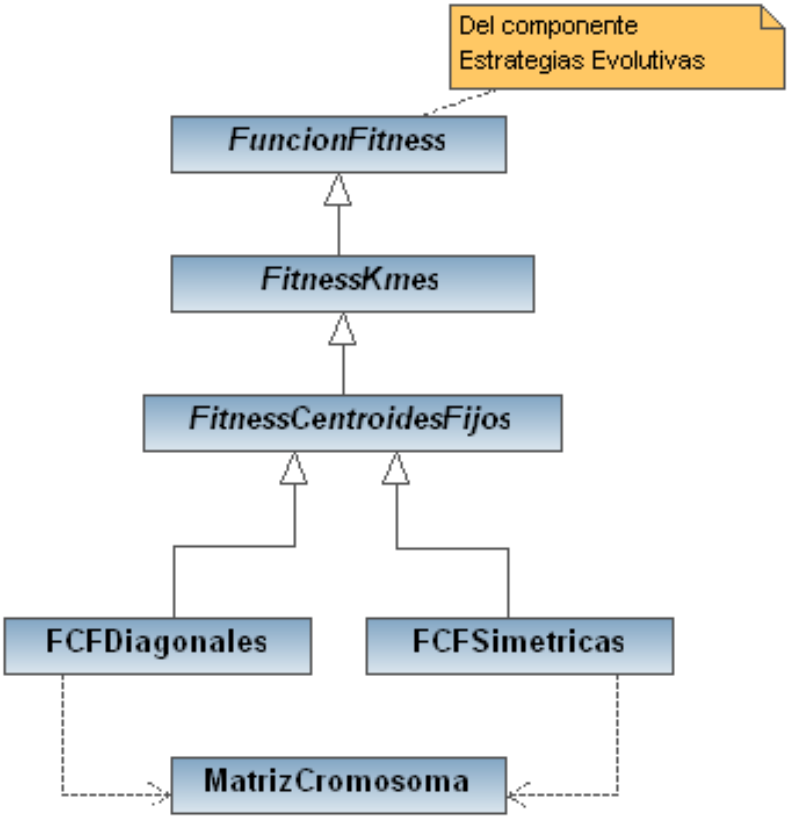


Figura 4.7: Subsistema KMES: funciones de fitness.

- Proporciona operaciones para codificar el cromosoma a partir de los elementos que vaya a contener.

La clase **FitnessCentroidesFijos** es la implementación de la función de evaluación correspondiente al método de clasificación utilizado. Esta clase se mantiene abstracta porque no se especifica la codificación y descodificación de una matriz (será distinta si las matrices son diagonales, simétricas, completas, etc.). Es en esta clase en la que se implementa la función *valueOf* (heredada de FunctionFitness), en la cual se devuelve el número de fallos de clasificación.

Las clases **FCFDiagonales** y **FCFSimetricas** extienden a la clase FitnessCentroides-Fijos, se trata de dos implementaciones de la misma función de fitness, para emplear con matrices diagonales y simétricas, respectivamente. En ambos casos se delega esta operación en la clase **MatrizCromosoma**, consiguiendo así facilitar la extensibilidad de la aplicación, ya que la codificación y descodificación de matrices podría ser empleada en

otra clase, si se decidiera añadir otra función de fitness que lo requiriera. En cualquier caso, para realizar estas operaciones se sigue lo especificado en las expresiones 4.2 y 4.3, en la página 36.

Capítulo 5

Experimentación

En este capítulo se describen los dominios que se han creado y elegido para llevar a cabo una experimentación con el método propuesto y se presentan los resultados obtenidos al experimentar con ellos.

Para ello, se explica cómo se ha generado u obtenido el dominio, qué características son las que lo hacen interesante en este contexto y cuáles son las propiedades que se espera que muestre el algoritmo. Una vez introducido el dominio, se explican qué pruebas se han realizado y los resultados obtenidos.

En algunos casos, se incluyen representaciones gráficas de funciones de distancia en dos dimensiones. Estas representaciones consisten en el trazado de puntos equidistantes, que formarán circunferencias o elipses. Se generan mediante GNU-Plot, a través de GNU-Octave; el fragmento de código de la figura 5.1, de GNU-Octave, permite realizar esta representación mediante la función *contour* y el desarrollo de la fórmula de la Distancia Euclídea Generalizada para dos dimensiones.

Además, se ha integrado en la aplicación desarrollada una pequeña utilidad para generar los dominios de datos, que permite especificar ciertos parámetros con el fin de permitir la generación de variantes de los dominios propuestos. En la sección A.7, *Generación de dominios de datos*, del Manual de usuario, se explica cómo utilizar dicha utilidad.

Para organizar los resultados presentados en esta sección se va a seguir la siguiente nomenclatura: cada uno de los experimentos está etiquetado como *Prueba X.Y*, donde la X corresponde al número del dominio en el que se ha ejecutado el experimento y la Y es un número secuencial que permite diferenciar entre las distintas pruebas realizadas.

```
function [X Y Z] = plotDistancia( M, c, ex, ey, es )
% M es la matriz de distancia
% c es el centro en el que se representa
% ex define el area en el eje x (desde el centro)
% ey define el area en el eje y (desde el centro)
% es define cuantas lineas se representan
[X Y] = meshgrid(c(1,1)-ex:es:c(1,1)+ex,c(1,2)-ex:es:c(1,2)+ex);
X2 = c(1,1) - X;
Y2 = c(1,2) - Y;
Z = sqrt(M(1,1)*X2.^2 + 2*X2.*Y2.*(M(1,2)+M(2,1)) + M(2,2)*Y2.^2);
contour(X,Y,Z);
endfunction
```

Figura 5.1: Código para la representación de funciones de distancia.

En todos los casos se experimenta utilizando validación cruzada con 10 subconjuntos (mostrándose la tasa de aciertos de validación), el resto de parámetros se configura según las características del dominio.

5.1. Dominio 1: nubes alineadas

Este dominio se genera para mostrar que, en algunos dominios, es necesario utilizar una función de distancia diferente a la Distancia Euclídea.

Consiste en cuatro "nubes de puntos", dispuestas como se muestra en la figura 5.2. Cada nube contiene el mismo número de puntos, y las que están por encima de 1,5 pertenecen a la clase *rojo*, mientras que las que están por debajo pertenecen a la clase azul.

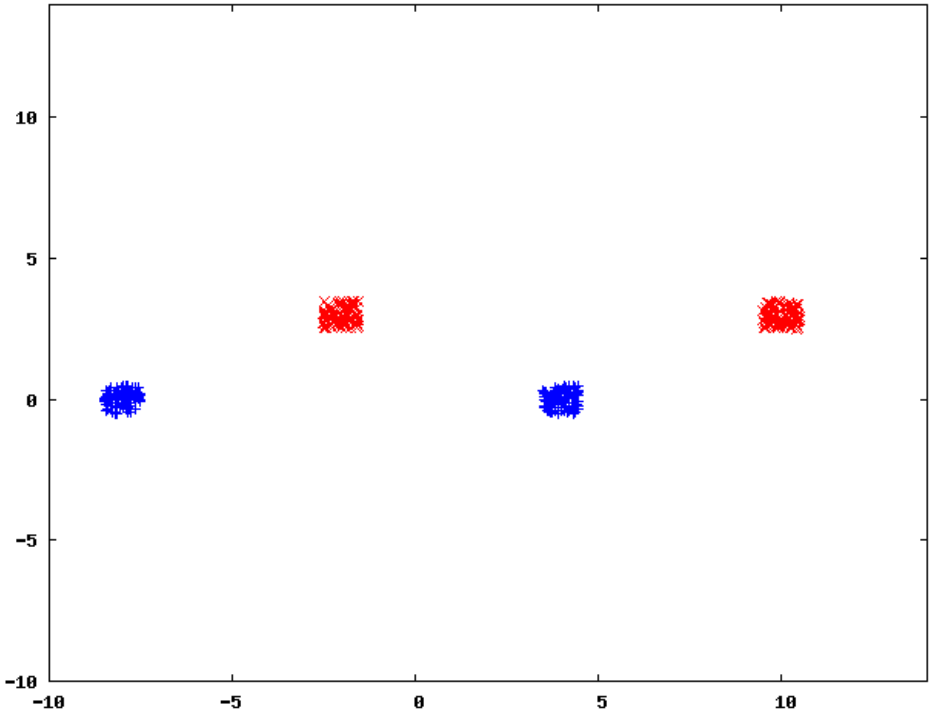


Figura 5.2: Representación gráfica del dominio *nubes alineadas*.

Empleando un solo centro por clase, con K-Medias, los valores que se obtendrían como centros serían la media de los puntos de cada clase. Estos puntos han sido representados en la figura 5.3.

Es fácil deducir que se obtendrá un 50 % de aciertos como resultado de clasificación utilizando la Distancia Euclídea, pues el centro de la clase azul es el más cercano a una de las nubes de puntos de la clase roja, y el centro de la clase roja es el más cercano a una de las nubes de la clase azul. Este hecho se aprecia mejor en la figura 5.4, en la que se representa la función de Distancia Euclídea desde cada uno de los centros.

Una tasa de aciertos del 50 % indica que los resultados del clasificador son bastante malos,

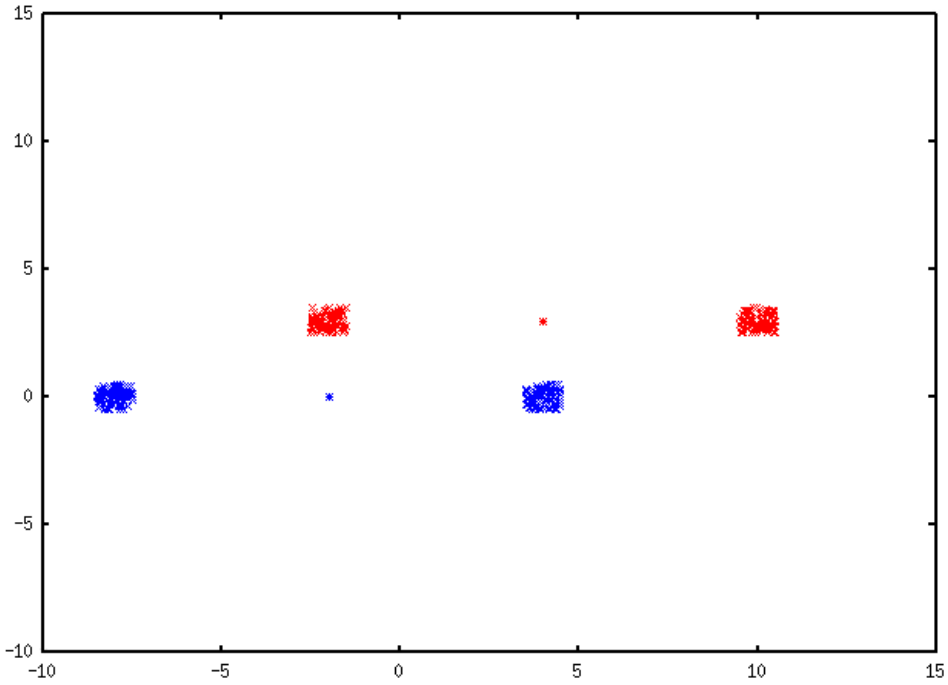


Figura 5.3: Centros en el dominio *nubes alineadas*.

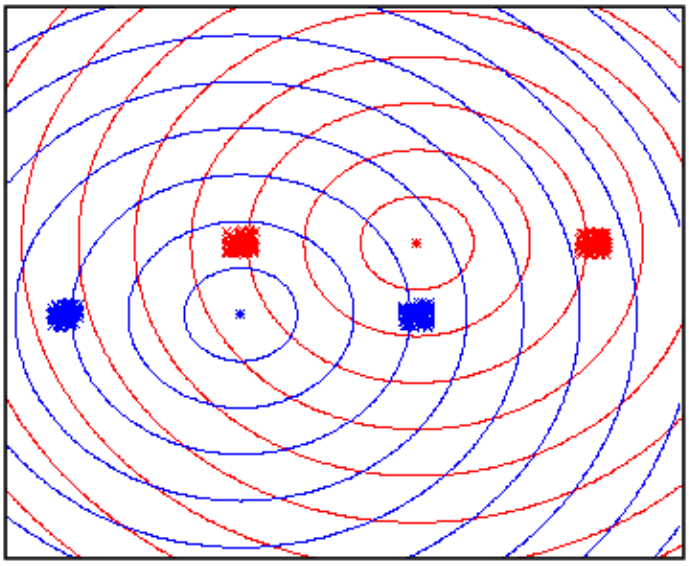


Figura 5.4: Representación de la Distancia Euclídea en el dominio *nubes alineadas*.

ya que, para este dominio, es la tasa de aciertos que cabría esperar de un clasificador aleatorio. Se espera, por tanto, que utilizando el método propuesto, se logre optimizar la Distancia Euclídea Generalizada para este dominio; ya que, de forma visual, se intuye que es necesario utilizar una Distancia Euclídea Ponderada, de forma que la representación de la función de distancia sea elíptica.

El dominio consta de 400 patrones, 200 de cada clase. Tal como están distribuidos los datos, es lógico utilizar un solo centroide por clase, que se situará como se pudo observar en la figura 5.3. Se emplean tanto matrices diagonales como simétricas, con el fin de observar las diferencias en los resultados. Por otra parte, para inicializar los centros al ejecutar K-Medias se emplea K-Medias++.

En cuanto a las técnicas a emplear, se definen tres: una de tipo CMA-ES, otra de tipo EE-(1+1) y una de estrategia evolutiva con poblaciones. En los tres casos se inicializan tanto los cromosomas como las varianzas con todos los valores a 1 (los cromosomas con un umbral de 0,5). Otros parámetros a destacar son los siguientes:

- 1000 como número máximo de generaciones.
- 0,00000001 como criterio de convergencia para las estrategias evolutivas simple y múltiples.
- Población de 20 individuos, cruce de 0,5 y torneo de 0,1 en las estrategias evolutivas múltiples.

En las tablas 5.1 y 5.2 (correspondientes a los pruebas 1.1 y 1.2) se presentan los datos obtenidos utilizando matrices simétricas y diagonales, respectivamente. En ambos casos se obtiene el 100 % de aciertos.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	100.00000	100.00000	100.00000
1	50.00000	100.00000	100.00000	100.00000
2	50.00000	100.00000	100.00000	100.00000
3	50.00000	100.00000	100.00000	100.00000
4	50.00000	100.00000	100.00000	100.00000
5	50.00000	100.00000	100.00000	100.00000
6	50.00000	100.00000	100.00000	100.00000
7	50.00000	100.00000	100.00000	100.00000
8	50.00000	100.00000	100.00000	100.00000
9	50.00000	100.00000	100.00000	100.00000
Media	**50.00**	**100.00**	**100.00**	**100.00**

Tabla 5.1: Resultados de la prueba 1.1.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	100.00000	100.00000	100.00000
1	50.00000	100.00000	100.00000	100.00000
2	50.00000	100.00000	100.00000	100.00000
3	50.00000	100.00000	100.00000	100.00000
4	50.00000	100.00000	100.00000	100.00000
5	50.00000	100.00000	100.00000	100.00000
6	50.00000	100.00000	100.00000	100.00000
7	50.00000	100.00000	100.00000	100.00000
8	50.00000	100.00000	100.00000	100.00000
9	50.00000	100.00000	100.00000	100.00000
Media	**50.00**	**100.00**	**100.00**	**100.00**

Tabla 5.2: Resultados de la prueba 1.2.

Los resultados obtenidos nos permiten confirmar que la optimización de la función de Distancia Euclídea Generalizada permite mejorar la tasa de aciertos para ciertos dominios. En este caso se ha pasado de unos resultados propios de un clasificador aleatorio a un 100 % de aciertos.

Se han obtenido los mismos resultados utilizando tanto matrices diagonales como matrices simétricas. Como cabía esperar, la diferencia entre utilizar un método u otro en este dominio está en el tiempo que ha tomado la evolución de las matrices.

En la tabla 5.3 se muestra el tiempo total de cómputo en segundos para ambos experimentos. Como cabía esperar, el esfuerzo computacional para hallar una matriz de distancia óptima en el caso del uso de matrices simétricas ha sido mayor que en el caso del uso de matrices diagonales, pues el tamaño del cromosoma era mayor.

Matriz	**cma**	**ee1**	**eem**
Diagonal	0.419	0.32	0.89
Simétrica	0.591	0.55	17.14

Tabla 5.3: Tiempo de ejecución para las pruebas 1.1 y 1.2.

Por último, se han obtenido las matrices y centros correspondientes tras realizar el entrenamiento con todos los datos del dominio, con el fin de comparar el clasificador obtenido con el correspondiente utilizando la Distancia Euclídea.

Se emplean los mismos parámetros de configuración que para el resto de experimentos, utilizando esta vez matrices diagonales con todos los valores no negativos. Por otra parte, sólo se emplea el algoritmo CMA-ES.

Se han obtenido los centros c_0 y c_1, y las matrices M_0 y M_1 para las clases azul y roja, respectivamente.

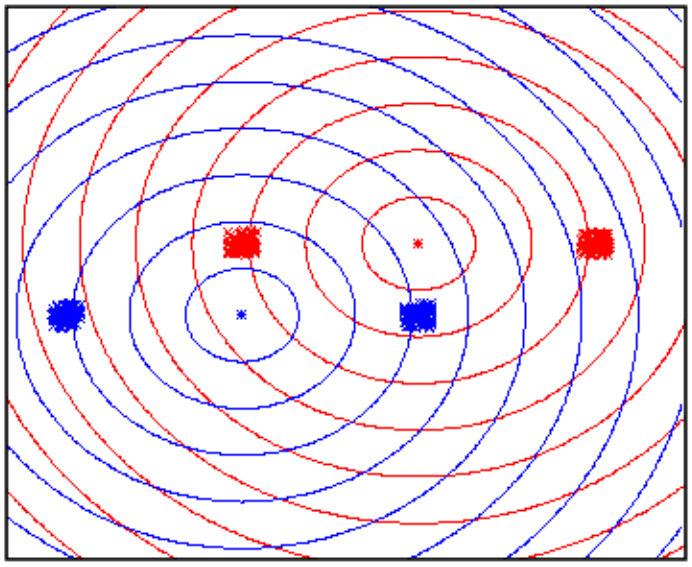
$$c_0 = \begin{bmatrix} \text{-}2.0043275 & 0.017383 \end{bmatrix}$$

$$c_1 = \begin{bmatrix} 4.0092792 & 2.977424 \end{bmatrix}$$

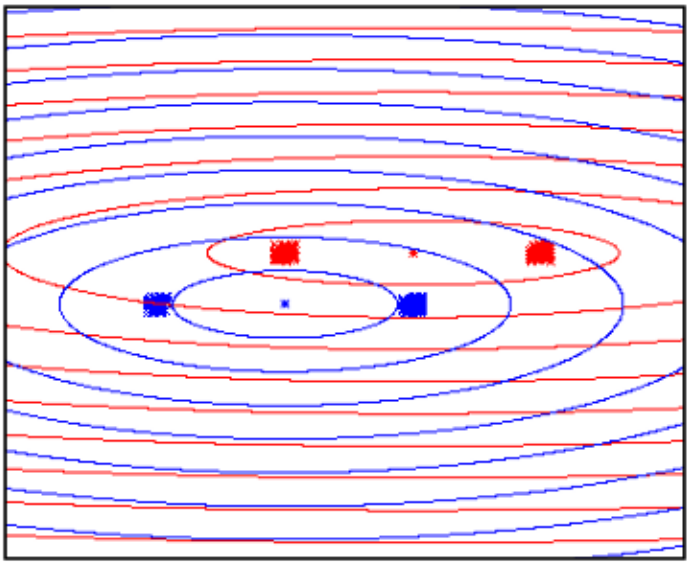
$$M_0 = \begin{bmatrix} 0.65423 & 0.0 \\ 0.0 & 4.91438 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 0.20060 & 0.0 \\ 0.0 & 0.50130 \end{bmatrix}$$

En la figura 5.5, al poder comparar las representaciones de las funciones de Distancia Euclídea y Euclídea Generalizada optimizada sobre los datos de este dominio, se aprecia claramente cómo se logra clasificar correctamente el 100 % de los datos mediante la optimización de la matriz de distancia. En la segunda figura, se representa la función de distancia correspondiente a la matriz M_0 centrada en c_0, y la correspondiente a la matriz M_1 centrada en c_1.



Representación de la Distancia Euclídea



Representación de las funciones GED optimizadas

Figura 5.5: Comparación entre la Distancia Euclídea y la GED optimizada.

5.2. Dominio 2: nubes reflejo

En el dominio anterior se trata de mostrar la necesidad de optimizar la función de distancia en dominios con ciertas características. En este dominio se va intentar mostrar la necesidad de encontrar una buena configuración de los parámetros del algoritmo. Se utiliza la misma idea que en el dominio *nubes alineadas*, pero en este caso van a ser necesarios 2 centroides por clase y una matriz por centroide para obtener buenos resultados.

Como la distribución de los puntos corresponde con la figura 5.2, la función de distancia a emplear para el centro que se situaría entre los puntos alineados horizontalmente de cada clase, debe ser distinta a la función de distancia a emplear con el que se situaría entre los centros alineados verticalmente. Por otra parte, una buena distribución inicial de los centros mediante el algoritmo K-Medias es imprescindible, por lo que es aconsejable utilizar un método de inicialización como el K-Medias++.

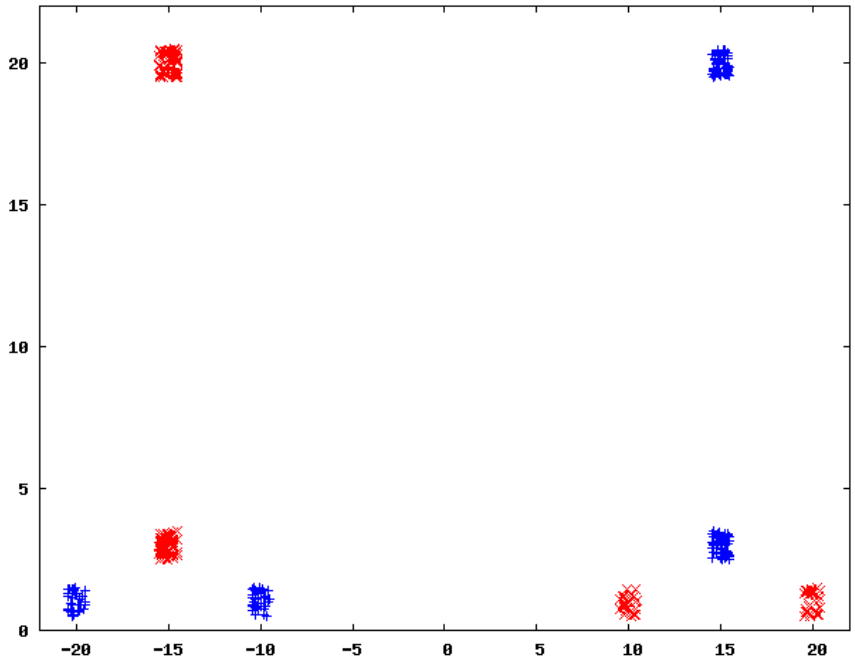


Figura 5.6: Representación gráfica del dominio *nubes reflejo*.

En este caso, se espera que en la clasificación utilizando la Distancia Euclídea se obtenga una tasa de 60 % de aciertos, pues el 40 % de los datos de cada clase, están concentrados en la nube de puntos situada en torno a la coordenada $y = 2, 5$. Al utilizar la función de Distancia Euclídea Generalizada optimizada mediante estrategias evolutivas esta tasa de

aciertos debería ser mejorada.

Este dominio está compuesto por 300 patrones, 150 de cada clase. Los experimentos que se van a realizar con él se van a plantear de forma diferente al caso anterior, en este caso se va a realizar una serie de ejecuciones para tratar de observar experimentalmente la importancia de encontrar una configuración apropiada para los distintos parámetros del algoritmo. El dominio *nubes reflejo* ha sido diseñado precisamente para explicar este hecho, por lo que algunos de los parámetros a utilizar pueden resultar muy evidentes. Pero en dominios reales se deberá atender a todo el conocimiento que se posea de los datos, que generalmente será insuficiente, por lo que los parámetros de configuración habrán de ajustarse experimentalmente.

Las distintas ejecuciones tienen en común la configuración de las técnicas de optimización de la matriz de distancia, que van a ser las mismas que en los experimentes realizados con el dominio 1, *nubes alineadas* (en la sección 5.1). Por otra parte, en los experimentos anteriores pudo observarse que para dominios en los que la Distancia Euclídea Ponderada es suficiente, es más adecuado utilizar matrices diagonales que simétricas; el proceso de optimización con éstas es más rápido. Por lo tanto, este parámetro también va a mantener para todos experimentos realizados con estos datos, utilizándose sólo matrices diagonales.

En primer lugar se va a atender al **número de centros** a utilizar. Se van emplean uno, dos, tres y cuatro centros por clase, cuyos resultados se muestran en las tablas 5.4, 5.5, 5.6 y 5.7 (que corresponden a las pruebas 2.1, 2.2, 2.3 y 2.4). Es importante señalar que en cada una de estas cuatro ejecuciones se va a emplear una matriz por centroide.

Con un solo centro se obtiene cierta mejora mediante los tres algoritmos empleados para optimizar la matriz de distancia, aunque no se alcanza el 100 % que se ha alcanzado al aumentar el número de centros. Observando la disposición de los centros representada en la figura 5.7 es fácil comprender por qué no se alcanzan mejores resultados, a pesar de que se logra ajustar la función de distancia para mejorar los resultados obtenidos mediante la Distancia Euclídea.

Cuando se han utilizado dos centros por clase, éstos se han situado en el lugar esperado (véase la figura 5.8) Mediante la optimización de la función de distancia se ha logrado pasar de una tasa de aciertos del 60 % a una del 100 %.

Para el uso de tres y cuatro centroides, lo ocurrido ha sido diferente. En la ejecución con tres centros por clase, también se ha obtenido el 100 % de aciertos al optimizar la función

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	60.00000	70.00000	78.14815	79.62963
1	60.00000	72.96296	77.77778	79.62963
2	60.00000	71.85185	80.00000	80.00000
3	60.00000	70.00000	80.00000	80.00000
4	60.00000	70.00000	74.07407	80.00000
5	60.00000	70.00000	80.00000	79.62963
6	60.00000	70.00000	80.00000	80.00000
7	60.00000	62.22222	80.00000	80.00000
8	60.00000	70.00000	75.55556	80.00000
9	60.00000	70.00000	80.00000	80.00000
Media	**60.00**	**70.00**	**77.33333**	**80.00**

Tabla 5.4: Resultados de la prueba 2.1.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	60.00000	100.00000	100.00000	100.00000
1	60.00000	100.00000	100.00000	100.00000
2	61.11111	100.00000	100.00000	99.25926
3	60.00000	100.00000	100.00000	100.00000
4	60.00000	100.00000	100.00000	100.00000
5	60.00000	100.00000	100.00000	100.00000
6	60.00000	100.00000	100.00000	100.00000
7	60.00000	100.00000	100.00000	100.00000
8	60.00000	100.00000	100.00000	100.00000
9	60.00000	100.00000	100.00000	100.00000
Media	**60.33333**	**100.00**	**100.00**	**100.00**

Tabla 5.5: Resultados de la prueba 2.2.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	80.00000	100.00000	100.00000	100.00000
1	80.00000	100.00000	100.00000	100.00000
2	80.00000	100.00000	100.00000	100.00000
3	80.00000	100.00000	100.00000	100.00000
4	100.00000	100.00000	100.00000	100.00000
5	100.00000	100.00000	100.00000	100.00000
6	100.00000	100.00000	100.00000	100.00000
7	100.00000	100.00000	100.00000	100.00000
8	100.00000	100.00000	100.00000	100.00000
9	100.00000	100.00000	100.00000	100.00000
Media	**92.00**	**100.00**	**100.00**	**100.00**

Tabla 5.6: Resultados de la prueba 2.3.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	100.00000	100.00000	100.00000	100.00000
1	100.00000	100.00000	100.00000	100.00000
2	100.00000	100.00000	100.00000	100.00000
3	100.00000	100.00000	100.00000	100.00000
4	100.00000	100.00000	100.00000	100.00000
5	100.00000	100.00000	100.00000	100.00000
6	100.00000	100.00000	100.00000	100.00000
7	100.00000	100.00000	100.00000	100.00000
8	100.00000	100.00000	100.00000	100.00000
9	100.00000	100.00000	100.00000	100.00000
Media	**100.00**	**100.00**	**100.00**	**100.00**

Tabla 5.7: Resultados de la prueba 2.4.

de distancia. Pero en este caso, la mejora respecto a la clasificación mediante la Distancia Euclídea no ha sido tan acusada. La situación de los centros, tal como se muestra en la figura 5.9, ha permitido que mediante la Distancia Euclídea ya se consiga un 92 % de aciertos.

Lo que se empieza a apreciar en el caso de tres centros, se confirma al utilizar cuatro por clase. En este caso, cada centro se ha situado en una de las nubes de puntos (figura 5.10), permitiendo que todos los datos se clasifiquen correctamente mediante la Distancia Euclídea.

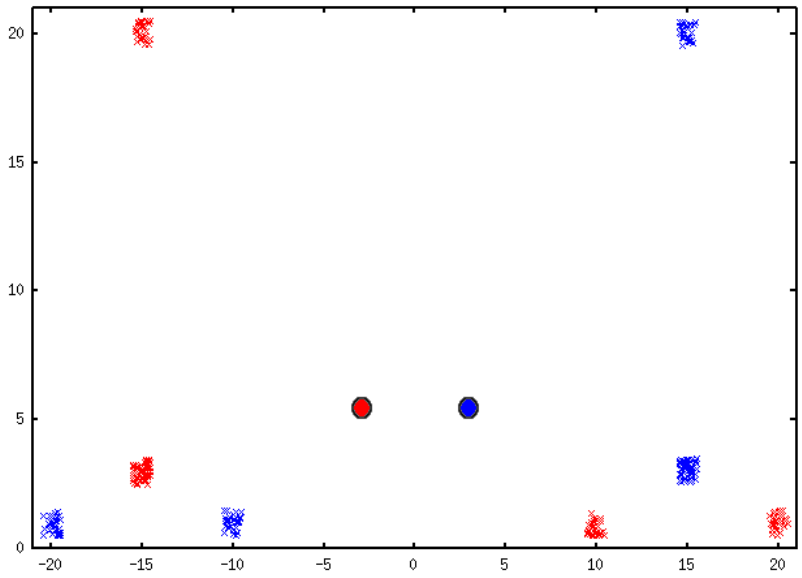


Figura 5.7: Disposición de 1 centro por clase para el dominio 2.

La mejora respecto a los resultados con la Distancia Euclídea ha sido mayor para el caso de 2 centros por clase. Por otra parte, al consistir la clasificación en encontrar el centro más cercano a un punto dado, el clasificador es más rápido cuanto menor sea el número de centros que lo formen. Por lo tanto, en los próximos experimentos se van a emplear dos centros por clase.

A continuación, se va a tratar el parámetro ***matriz.por***, dando algunas razones para elegir un valor determinado y explicando las particularidades de este dominio concreto respecto a dicho parámetro.

Generalmente es más eficiente utilizar la misma matriz para todos los datos de cada clase, al ser menor el número de matrices a evolucionar (reduciéndose por tanto el tamaño del cromosoma). Además, parece lógico pensar que aplicando la misma transformación a todos los datos de la misma clase, el resultado de la clasificación mejorará, pues lo más probable

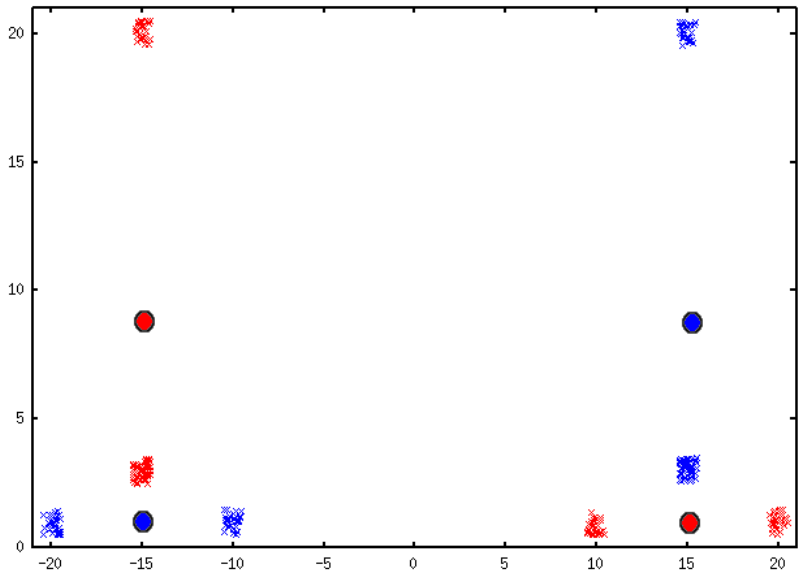


Figura 5.8: Disposición de 2 centros por clase para el dominio 2.

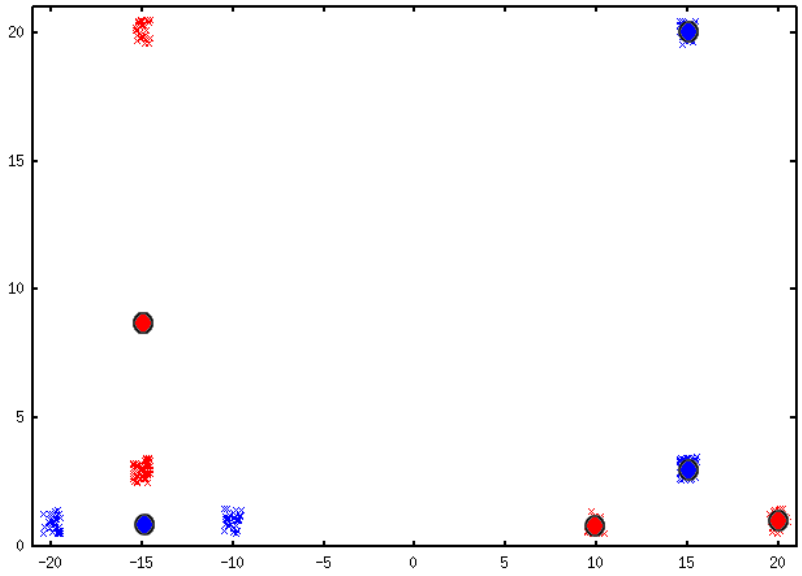


Figura 5.9: Disposición de 3 centros por clase para el dominio 2.

es que todos estén dispuestos de la misma forma.

Por el contrario, tal como se puede observar en los resultados presentados en la tabla 5.8 (prueba 2.5), en este caso, a pesar de obtenerse cierta mejora respecto a los resultados obtenidos con la Distancia Euclídea, no se logra alcanzar el 100 % como al utilizar una función de distancia por cada centro (resultados en la tabla 5.5). Los centroides se disponen inicialmente utilizando la Distancia Euclídea, por lo que su disposición es la misma en los dos casos (la representada en la figura 5.8). Pero para poder clasificar correctamente

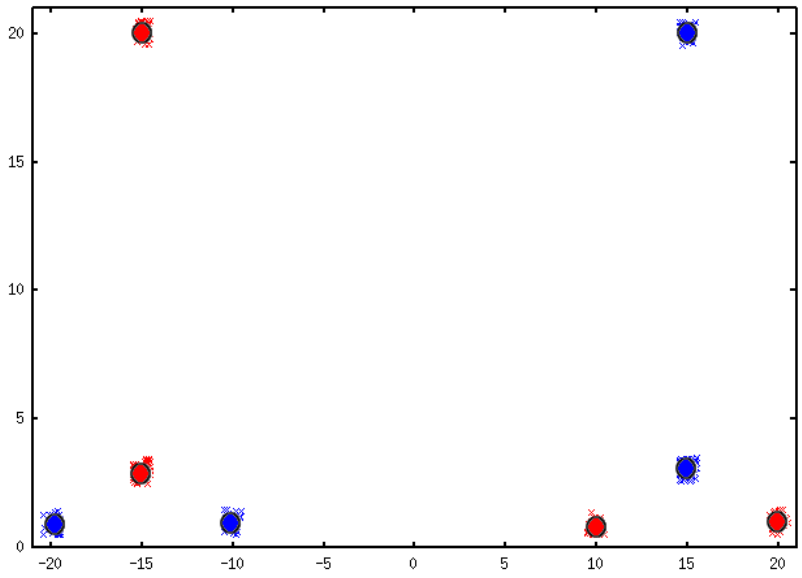


Figura 5.10: Disposición de 4 centros por clase para el dominio 2.

todos los datos, la función de distancia no debe ser la misma para los dos centros de cada clase.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	61.48148	76.66667	84.07407	90.00000
1	60.00000	80.00000	84.81481	80.00000
2	60.00000	80.00000	90.00000	80.00000
3	60.00000	80.00000	83.33333	90.00000
4	60.00000	80.00000	90.00000	76.29630
5	60.00000	80.00000	80.00000	80.00000
6	60.00000	90.00000	84.81481	80.00000
7	60.00000	80.00000	80.00000	80.00000
8	60.00000	80.00000	80.00000	84.44444
9	60.00000	80.00000	80.00000	89.25926
Media	**60.00**	**80.66667**	**83.33333**	**82.66667**

Tabla 5.8: Resultados de la prueba 2.5.

5.3. Dominio 3: nubes rotadas

Con este dominio se pretende investigar qué ocurre al intentar realizar la clasificación en datos en los que se han realizado rotaciones.

Para ello se ha tomado el dominio *nubes alineadas*, para el que se han obtenido rápidamente resultados satisfactorios, y se le ha aplicado una rotación de 45 grados, obteniendo así el dominio *nubes rotadas*. Como el dominio en el que se basa, se trata de un conjunto de datos sencillo, para el que es necesario un único centro por clase y para el cual se espera una tasa de aciertos del 50 % utilizando la Distancia Euclídea. En la figura 5.11 se puede observar cómo están dispuestos los datos.

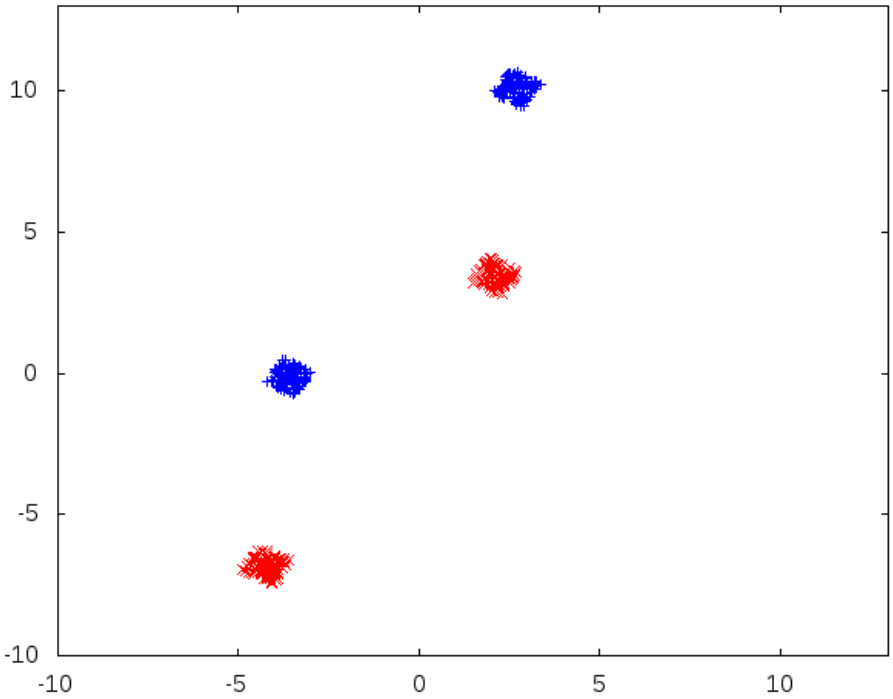


Figura 5.11: Representación gráfica del dominio *nubes rotadas*.

Al observar la disposición de los datos, parece que la optimización de matrices de distancia diagonales no será suficiente para clasificar correctamente el 100 % de los datos. Pero se van a ejecutar experimentos utilizando matrices simétricas y diagonales, para poder comparar los resultados obtenidos. Para ello se emplea la misma configuración que los experimentos realizados con el dominio 1. Las tablas 5.9 y 5.10 (correspondientes a las pruebas 3.1 y 3.2) muestran los resultados obtenidos para matrices diagonales y simétricas respectivamente.

Se observa que en ambos casos se obtienen buenos resultados, aunque utilizando matrices

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	100.00000	100.00000	100.00000
1	50.00000	100.00000	100.00000	89.44444
2	50.00000	100.00000	100.00000	100.00000
3	50.00000	100.00000	100.00000	100.00000
4	50.00000	100.00000	100.00000	100.00000
5	50.00000	100.00000	100.00000	100.00000
6	50.00000	100.00000	100.00000	100.00000
7	50.00000	100.00000	100.00000	100.00000
8	50.00000	100.00000	100.00000	100.00000
9	50.00000	90.00000	100.00000	100.00000
Media	**50.00**	**98.75**	**100.00**	**99.00**

Tabla 5.9: Resultados de la prueba 3.1.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	100.00000	100.00000	100.00000
1	50.00000	100.00000	100.00000	100.00000
2	50.00000	100.00000	100.00000	100.00000
3	50.00000	100.00000	100.00000	100.00000
4	50.00000	100.00000	100.00000	100.00000
5	50.00000	100.00000	100.00000	100.00000
6	50.00000	100.00000	100.00000	100.00000
7	50.00000	100.00000	100.00000	100.00000
8	50.00000	100.00000	100.00000	100.00000
9	50.00000	100.00000	100.00000	100.00000
Media	**50.00**	**100.00**	**100.00**	**100.00**

Tabla 5.10: Resultados de la prueba 3.2.

diagonales no siempre se alcanza el 100 % de aciertos. Además, tal como se muestra en la tabla 5.11, se ha necesitado mucho más tiempo para optimizar las matrices diagonales que para optimizar las simétricas, cuando en el caso de las simétricas el cromosoma está compuesto por más elementos.

Resulta extraño el hecho de obtener tan buenos resultados con matrices diagonales en este dominio, en el cual se ha aplicado una rotación. Este hecho, probablemente, se debe a

Matriz	**cma**	**ee1**	**eem**
Diagonal	1.248	0.414	18.533
Simétrica	0.327	0.098	0.646

Tabla 5.11: Tiempo de ejecución en el dominio 3 con matrices generales.

considerar infinitas las distancias cuyo cuadrado es negativo. Para comprobar si esto tiene algún efecto, se realizan experimentos utilizando sólo matrices con valores no negativos, cuyos resultados se muestran en las tablas 5.12 y 5.13.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	75.00000	75.00000	75.00000
1	50.00000	75.00000	75.00000	75.00000
2	50.00000	75.00000	75.00000	75.00000
3	50.00000	75.00000	75.00000	75.00000
4	50.00000	75.00000	75.00000	75.00000
5	50.00000	75.00000	75.00000	75.00000
6	50.00000	75.00000	75.00000	75.00000
7	50.00000	75.00000	75.00000	75.00000
8	50.00000	75.00000	75.00000	75.00000
9	50.00000	75.00000	75.00000	75.00000
Media	**50.00**	**75.00**	**74.75**	**75.00**

Tabla 5.12: Resultados de la prueba 3.3.

Utilizando matrices diagonales con todos los valores no negativos no ha sido posible obtener una tasa del 100 % de aciertos, aunque encontramos cierta mejora respecto a utilizar la Distancia Euclídea. Por el contrario, al utilizar matrices de distancias simétricas se ha podido clasificar correctamente el total de los datos.

Por otro lado, puede resultar sorprendente que mediante una matriz simétrica para la cual todos los valores son no negativos sea posible clasificar el 100 % de los datos de un dominio en el que los datos se han proyectado mediante una matriz de rotación (se ha aplicado rotación mediante la matriz R con $\alpha = 45^{\text{o}}$).

$$R = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	100.00000	100.00000	100.00000
1	50.00000	100.00000	100.00000	100.00000
2	50.00000	100.00000	100.00000	100.00000
3	50.00000	100.00000	100.00000	100.00000
4	50.00000	100.00000	100.00000	100.00000
5	50.00000	100.00000	100.00000	100.00000
6	50.00000	100.00000	100.00000	100.00000
7	50.00000	100.00000	100.00000	100.00000
8	50.00000	100.00000	100.00000	100.00000
9	50.00000	100.00000	100.00000	100.00000
Media	**50.00**	**100.00**	**100.00**	**100.00**

Tabla 5.13: Resultados de la prueba 3.4.

Pero si recordamos que la matriz que se ha optimizado es $M = M_D \cdot M_D^T$, apreciamos que no es necesario que esta matriz contenga valores negativos para que incluya la transformación correspondiente a la rotación.

Si la matriz M correspondía a una función de Distancia Euclídea Generalizada que permitía clasificar correctamente todos los datos antes de aplicarle la rotación, la función de distancia correspondiente a la matriz $M_R = R\ M_D\ M_D^T\ R^T$, permitirá clasificar correctamente los datos tras aplicarles la rotación; tal como se muestra en la ecuación 5.1.

$$d(xR, yR)_M = \sqrt{(xR - yr)M_D\ M_D^T(xR - yR)^T} = \sqrt{(x - y)\ R\ M_D\ M_D^T\ R^T\ (x - y)^T)} \tag{5.1}$$

Suponiendo la matriz M diagonal, por simplicidad, y dado el hecho que es posible obtener una tasa de aciertos del 100 % utilizando matrices diagonales (tal como se pudo observar en la tabla 5.9). La matriz M_R quedaría definida según la expresión 5.2, siendo

$$M_D = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}.$$

$$M_R = R\ M_D\ M_D^T\ R^T = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} = \tag{5.2}$$

$$\begin{bmatrix} a^2\cos^2\alpha + b^2\sin^2\alpha & (b^2 - a^2)\sin\alpha\cos\alpha \\ (b^2 - a^2)\sin\alpha\cos\alpha & a^2\sin^2\alpha + b^2\cos^2\alpha \end{bmatrix}$$

De modo que ninguno de los elementos de la matriz M_R deba ser necesariamente negativo. Esto quiere decir que, a pesar de restringir las matrices utilizadas a aquellas que sólo tengan elementos no negativos se pueden obtener buenos resultados de clasificación en dominios para los que sean necesarias rotaciones.

5.4. Dominio 4: elipses

Tomado como base el dominio 2 *nubes reflejo*, se ha generado un dominio más realista, en el que los datos no están tan claramente colocados para que sea tan evidente la disposición lógica de los centros tras la ejecución inicial de K-Medias.

En este caso los datos se han distribuido de forma elíptica, habiendo una mayor concentración de puntos en cada uno de los extremos de la diagonal mayor. En total hay 536 datos, 268 de cada clase. No se esperan buenos resultados utilizando la Distancia Euclídea, esperando mejorarlos optimizando la función de Distancia Euclídea Generalizada optimizada. En la figura 5.12 se muestra una representación gráfica de la disposición de los datos.

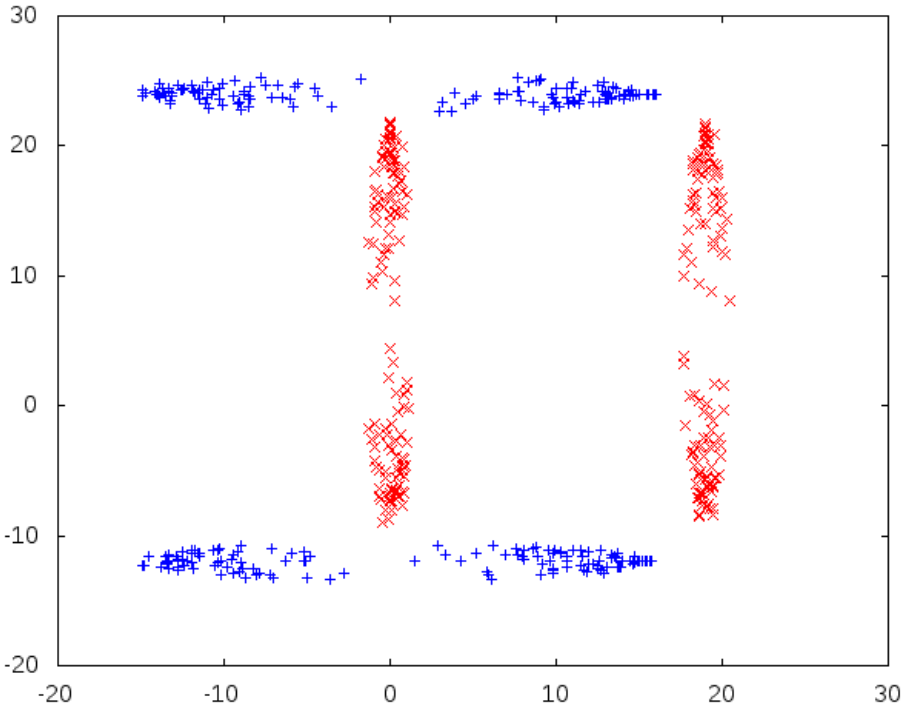


Figura 5.12: Representación gráfica del dominio *nubes rotadas*.

Utilizando dos centros por clase y la configuración que se ha empleado en dominios anteriores se obtienen los resultados presentados en la tabla 5.14 utilizando matrices diagonales y los presentados en la tabla 5.15 para matrices simétricas.

Tanto utilizando matrices simétricas como matrices diagonales, se obtiene una mejora sustancial respecto a los resultados obtenidos mediante la Distancia Euclídea. Llama la atención que la tasa de aciertos de clasificación obtenida mediante la Distancia Euclídea

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	86.79245	100.00000	100.00000	100.00000
1	77.35849	100.00000	100.00000	98.11321
2	56.60377	98.11321	98.11321	98.11321
3	90.56604	100.00000	96.22642	100.00000
4	84.90566	100.00000	98.11321	96.22642
5	66.03774	100.00000	100.00000	81.13208
6	58.49057	100.00000	98.11321	94.33962
7	66.03774	98.11321	98.11321	100.00000
8	66.03774	100.00000	92.45283	98.11321
9	83.05085	100.00000	100.00000	100.00000
Media	**73.69403**	**99.62687**	**98.13433**	**96.64179**

Tabla 5.14: Resultados de la prueba 4.1.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	66.03774	86.79245	94.33962	94.33962
1	77.35849	100.00000	100.00000	100.00000
2	83.01887	98.11321	100.00000	100.00000
3	58.49057	100.00000	98.11321	90.56604
4	84.90566	100.00000	98.11321	98.11321
5	92.45283	100.00000	100.00000	96.22642
6	58.49057	94.33962	86.79245	81.13208
7	62.26415	96.22642	92.45283	98.11321
8	66.03774	100.00000	94.33962	98.11321
9	66.10169	100.00000	91.52542	100.00000
Media	**71.45522**	**97.57463**	**95.52239**	**95.70896**

Tabla 5.15: Resultados de la prueba 4.2.

no sea la misma en las dos ejecuciones. Esto se debe, probablemente, a que la distribución inicial de los centros utilizando K-Medias ha sido distinta, favoreciendo ligeramente la clasificación en la ejecución en la que se han empleado matrices diagonales.

Otro aspecto a tener en cuenta es el tiempo de ejecución, en la tabla 5.16 se muestra, en segundos, el tiempo que ha llevado la optimización de la función de distancia en ambas ejecuciones. En el caso de las matrices simétricas, el tiempo de ejecución ha sido algo

mayor que en el caso de las matrices diagonales. Por otra el proceso de optimización utilizando estrategias evolutivas múltiples ha sido mucho más costoso en ambos casos.

Matriz	**cma**	**ee1**	**eem**
Diagonal	15.613	20.928	715.311
Simétrica	26.089	23.319	1248.133

Tabla 5.16: Tiempo de ejecución para el dominio 4.

5.5. Dominio 5: atributos aleatorios

Este tipo de dominios ya se ha empleado en otros trabajos [Valls, 2009] para comprobar si un método de clasificación es capaz de discriminar atributos irrelevantes. Cada una de las muestras tiene cuatro atributos, los dos primeros se generan aleatoriamente en el intervalo $[0, 1]$ y los dos últimos se generan aleatoriamente en el intervalo $[0, 100]$.

Sólo los dos primeros atributos son relevantes, pues se les asigna una clase u otra según la siguiente regla: si el primer atributo es mayor que el segundo, entonces pertenece a la clase 0; en cambio, si el primer atributo es menor o igual que el primero, entonces se le asigna la clase 1.

Se ejecutan varios experimentos, pero no se logran mejorar los resultados de clasificación con ninguna de las configuraciones probadas. De forma representativa, se incluyen en la tabla 5.17 los resultados de la prueba 5.1, en la que se emplea la configuración que se ha venido utilizando en los dominios anteriores y matrices diagonales.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	60.00000	30.00000	50.00000
1	30.00000	30.00000	30.00000	30.00000
2	70.00000	60.00000	40.00000	50.00000
3	40.00000	50.00000	40.00000	50.00000
4	60.00000	60.00000	50.00000	40.00000
5	60.00000	70.00000	60.00000	70.00000
6	50.00000	50.00000	50.00000	50.00000
7	30.00000	30.00000	30.00000	30.00000
8	60.00000	60.00000	60.00000	50.00000
9	70.00000	70.00000	40.00000	60.00000
Media	**52.00**	**54.00**	**43.00**	**48.00**

Tabla 5.17: Resultados de la prueba 5.1.

No se logra obtener mejores resultados mediante la optimización de las funciones de distancia por el siguiente motivo: el algoritmo K-Medias se ejecuta una única vez, al inicio, de forma que los centros se sitúan únicamente al principio. Al utilizar la Distancia Euclídea para situar los centros inicialmente, su distribución está muy influenciada por los atributos más significativos, que en este dominio son precisamente los atributos aleatorios.

Dado que los centros han sido situados en base a los atributos aleatorios y que su posición no varía durante todo el proceso de optimización de la función de distancia, resulta imposible que esta optimización permita mejorar la clasificación.

Por lo tanto el algoritmo propuesto no permite discriminar atributos irrelevantes de forma automática. Esta carencia detectada va a intentar solventarse mediante una variante del algoritmo propuesto.

5.6. Variante del método propuesto

Gracias a los resultados obtenidos con el dominio *atributos aleatorios* se ha podido detectar que el método propuesto no permite discriminar atributos irrelevantes para la clasificación.

Esta carencia detectada se trata de solucionar mediante la variante del método original que se propone en esta sección. En primer lugar se presentan los cambios que se van a introducir al algoritmo diseñado, posteriormente se explica cómo estos cambios son implementados, y por último, se presentan una serie de experimentos para comprobar el funcionamiento del nuevo algoritmo propuesto.

5.6.1. Cambios introducidos

Tal como se explicaba en la sección 5.5 (*Dominio 5: atributos aleatorios*), el problema a la hora de discriminar atributos irrelevantes en ciertos dominios es que los centros se mantienen fijos durante todo el proceso de optimización de la matriz. Entonces, se debe introducir algún mecanismo para volver a situar dichos centros durante el proceso de evolución.

Por otro lado, lo deseable para que, mediante la optimización de la función de distancia, no se tengan en cuenta ciertos atributos del dominio es que aparezcan una serie de ceros en la matriz. Por lo que se va a proveer al algoritmo de mecanismos para que sea más fácil que durante la evolución se incluyan ceros en la matriz.

Con estos objetivos se van a incluir los siguientes cambios:

1. Los centros se vuelven a distribuir mediante K-Medias cada vez que se calcula el fitness. Lógicamente, K-Medias se ejecuta utilizando las funciones de distancia correspondientes a las matrices que se están evolucionando. Dado que corresponden a funciones de distancia cuyas propiedades están relajadas, es probable que no se alcance la convergencia con el algoritmo K-Medias. Para solventar esto se proponen dos soluciones:

    - Si el algoritmo K-Medias no converge en menos de 10000 iteraciones se penaliza al individuo que se está evaluando, se establece como fitness el valor máximo

posible[1].

- Generalmente se recomienda utilizar matrices con todos los elementos no negativos, pues se garantiza que son definidas positivas y que se cumplen las propiedades de las funciones de distancia.

2. Se cambia la forma de calcular el valor de adecuación con el objetivo de *premiar* a los individuos para los que las matrices contengan ceros o valores próximos a cero, además es interesante que si se clasifican correctamente todos los datos, el fitness sea óptimo, o lo que es lo mismo, que su valor sea cero. Para ello el *fitness* se calcula según la ecuación 5.3.

$$F = f \cdot (\sum_{c_i \in C} |c_i| + \sum_{c_i \in C} z(c_i) + f) \tag{5.3}$$

donde:

F es el fitness a calcular.

f es el número de fallos en la clasificación.

c_i es cada elemento del conjunto C.

La función z se define de la forma siguiente:

$$z(c_i) = \begin{cases} 0, \text{ si } c_i = 0 \\ \\ 1, \text{ si } c_i \neq 0 \end{cases}$$

3. En las estrategias evolutivas simples y múltiples se introduce un nuevo operador que pone a cero un elemento del cromosoma con cierta probabilidad, la cual será determinada como parámetro.

Al introducir estos cambios en el algoritmo propuesto, se espera dotarle de la capacidad de discriminar los atributos irrelevantes a la hora de realizar la clasificación. Ahora el algoritmo K-Medias se ejecuta cada vez que se calcula el valor de adecuación de un individuo, por lo que la ejecución de esta variante del algoritmo original será mucho más lenta.

[1]El valor máximo representable con el tipo *Double* de Java (*Double.MAX_VALUE*).

5.6.2. Implementación

Para implementar los cambios introducidos ha sido necesario modificar dos subsistemas: el subsistema *Estrategias Evolutivas* y el subsistema *KMES*.

Las modificaciones en el subsistema Estrategias Evolutivas han resultado sencillas, simplemente había que introducir el operador genético que, en cada iteración, permitiera poner un valor del cromosoma a cero con cierta probabilidad. Estos cambios se introducen en las clases **EESimple** y **EEMultiple**.

El resto de cambios se han realizado sobre el subsistema KMES. En primer lugar, se modifica la clase **Configuracion**, para que los cambios introducidos sean configurables en el fichero de configuración (no perdiéndose la versión original del algoritmo), y **Kmes** para que estas configuraciones sean aplicadas.

La implementación del cálculo del *fitness* de los individuos ha consistido en añadir tres nuevas clases: la clase **FitnessKMedias**, en la que se implementa el cálculo del fitness según la fórmula 5.3 tras la redistribución de los centros utilizando K-Medias con las funciones de distancia correspondientes, y las clases **FKMDiagonales** y **FKMSimetricas**, para el uso de matrices diagonales y simétricas, respectivamente.

En la figura 5.13 se puede observar la estructura de las clases involucradas en el cálculo del *fitness* de la aplicación, tras haber introducido los cambios relativos a esta nueva variante.

5.6.3. Experimentación

Se van a realizar una serie de ejecuciones con el dominio 5, *atributos aleatorios* especificado en la sección 5.5, ya que los resultados en este dominio son los que han motivado el desarrollo de esta variante del algoritmo.

En la tabla 5.18 se presentan los resultados obtenidos utilizando matrices diagonales, empleándose la configuración utilizada en el resto de experimentos, salvo por los cambios que se indican a continuación:

- Se utiliza el método *k-Medias* para el cálculo del *fitness*, en lugar del método

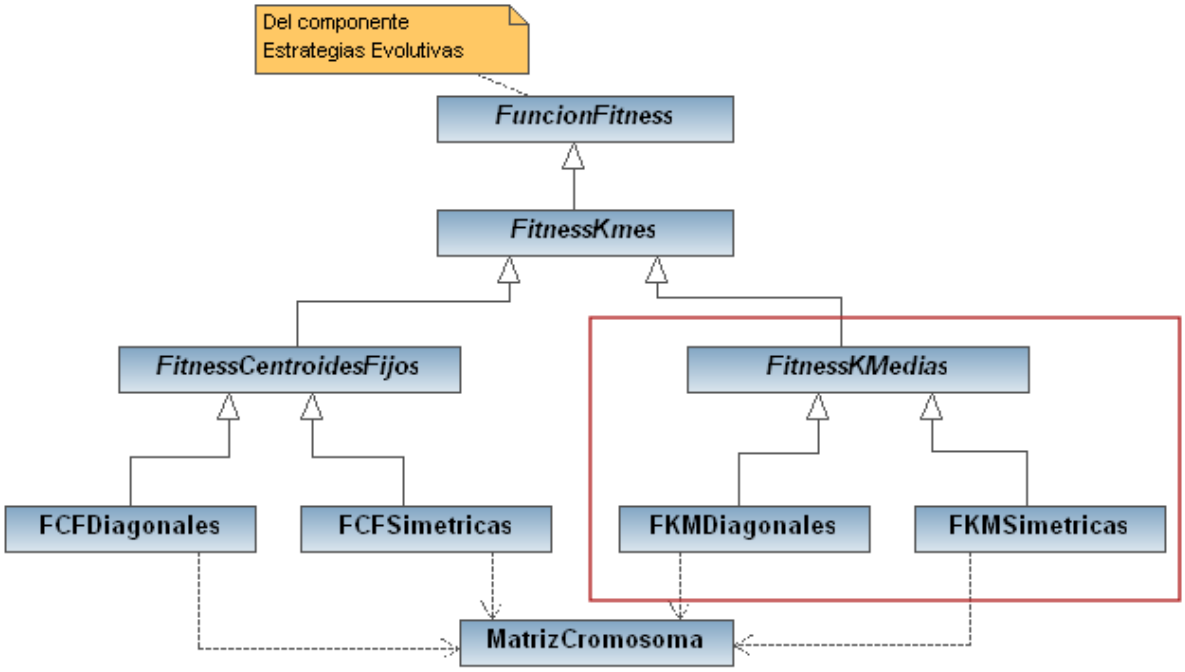


Figura 5.13: Subsistema KMES: cambios en las funciones de fitness.

centroides-fijos.

- Sólo se emplean matrices con valores no negativos.
- Se utiliza el nuevo operador genético en las estrategias evolutivas simples y múltiples con probabilidad 0,1.

Los resultados obtenidos no son muy prometedores, pues tampoco se han logrado mejoras en este caso. Así que se ha repetido la ejecución aumentado el número de iteraciones máximo para todas las técnicas de optimización de la función de distancia; este valor ha pasado de 1000 a 5000. Los resultados se presentan en la tabla 5.19.

En este caso se ha logrado mejorar los resultados de clasificación respecto a los obtenidos mediante la Distancia Euclídea. Las estrategias evolutivas múltiples han resultado ser la técnica que mejor ha logrado ajustar las matrices de distancia, también han sido el método que más tiempo de cómputo ha requerido, tal como se muestra en la tabla 5.20. Parece que se requería realizar una exploración de las soluciones que las estrategias evolutivas simples no podían alcanzar con estas características.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	50.00000	60.00000	50.00000	40.00000
1	30.00000	20.00000	30.00000	30.00000
2	70.00000	50.00000	50.00000	60.00000
3	40.00000	40.00000	40.00000	90.00000
4	60.00000	60.00000	60.00000	50.00000
5	60.00000	60.00000	50.00000	30.00000
6	50.00000	50.00000	70.00000	60.00000
7	30.00000	30.00000	30.00000	30.00000
8	60.00000	60.00000	40.00000	60.00000
9	70.00000	50.00000	70.00000	50.00000
Media	**52.00**	**48.00**	**49.00**	**50.00**

Tabla 5.18: Resultados de la prueba 5.2.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	54.44444	100.00000	100.00000	93.33333
1	56.66667	100.00000	55.55556	100.00000
2	53.33333	100.00000	48.88889	96.66667
3	55.55556	100.00000	56.66667	96.66667
4	56.66667	100.00000	100.00000	97.77778
5	55.55556	100.00000	100.00000	100.00000
6	54.44444	60.00000	54.44444	96.66667
7	57.77778	98.88889	57.77778	88.88889
8	54.44444	100.00000	50.00000	98.88889
9	54.44444	62.22222	52.22222	96.66667
Media	**52.00**	**89.00**	**61.00**	**97.00**

Tabla 5.19: Resultados de la prueba 5.3

Matriz	**cma**	**ee1**	**eem**
Diagonal	183.801	58.50	2542.216

Tabla 5.20: Tiempo de ejecución de la prueba 5.3.

Se espera poder mejorar aún más los resultados utilizando el algoritmo CMA-ES para optimizar la matriz de distancia, por lo que se ha realizado una ejecución aumentando el

número de iteraciones hasta 100000, los resultados se incluyen en la tabla 5.21.

Subconjunto	**Euclídea**	**cma**
0	54.44444	100.00000
1	56.66667	100.00000
2	53.33333	100.00000
3	55.55556	100.00000
4	56.66667	100.00000
5	55.55556	100.00000
6	54.44444	100.00000
7	57.77778	100.00000
8	54.44444	100.00000
9	54.44444	96.66667
Media	**52.00**	**97.00**

Tabla 5.21: Resultados de la prueba 5.4.

Utilizando el algoritmo CMA-ES con 10000 iteraciones máximas se ha logrado alcanzar una tasa de aciertos en la clasificación del 97 %, durando la ejecución 231.367 segundos, aproximadamente 10 veces menos que las estrategias evolutivas múltiples en la ejecución anterior.

Para probar el efecto del nuevo operador genético introducido en las estrategias evolutivas simples y múltiples se va a aumentar a 0,5 la probabilidad de poner a cero un valor del cromosoma en cada generación. Se realiza una nueva ejecución utilizando estas dos técnicas para optimizar la matriz de distancia, habiendo introducido este cambio en la figuración y fijando a 5000 el número máximo de iteraciones; la tabla 5.22 incluye los resultados.

La mejora obtenida ha sido notable. En ambos casos se ha alcanzado una tasa del 100 % y se ha convergido a la solución realmente rápido, tal como se muestra en la tabla 5.23.

Por último se realiza una ejecución utilizando matrices simétricas, utilizando 10000 iteraciones máximas para el algoritmo CMA-ES y 5000 para las estrategias evolutivas, además de probabilidad 0,5 para el nuevo operador genético. Los resultados se muestran en la tabla 5.24.

En este caso sólo se han mejorado de forma significativa los resultados de la clasificación mediante la técnica CMA-ES, y no se han logrado tasas de aciertos próximas al 100 %.

Subconjunto	**Euclídea**	**ee1**	**eem**
0	54.44444	100.00000	100.00000
1	56.66667	100.00000	100.00000
2	53.33333	100.00000	100.00000
3	55.55556	100.00000	100.00000
4	56.66667	100.00000	100.00000
5	55.55556	100.00000	100.00000
6	54.44444	100.00000	100.00000
7	57.77778	100.00000	100.00000
8	54.44444	100.00000	100.00000
9	54.44444	100.00000	100.00000
Media	**52.00**	**100.00**	**100.00**

Tabla 5.22: Resultados de la prueba 5.5.

Matriz	**ee1**	**eem**
Diagonal	4.211	32.984

Tabla 5.23: Tiempo de ejecución de la prueba 5.5.

Subconjunto	**Euclídea**	**cma**	**ee1**	**eem**
0	54.44444	95.55556	53.33333	96.66667
1	56.66667	90.00000	50.00000	100.00000
2	53.33333	84.44444	54.44444	54.44444
3	55.55556	87.77778	52.22222	96.66667
4	56.66667	74.44444	45.55556	58.88889
5	55.55556	68.88889	51.11111	57.77778
6	54.44444	72.22222	54.44444	57.77778
7	57.77778	84.44444	53.33333	58.88889
8	54.44444	75.55556	50.00000	50.00000
9	54.44444	75.55556	50.00000	68.88889
Media	**52.00**	**70.00**	**55.00**	**63.00**

Tabla 5.24: Resultados de la prueba 5.6.

5.7. Comparativa entre los métodos propuestos

En esta sección se cotejan las características de los dos algoritmos de clasificación propuestos en este proyecto. Para ello se resaltan las diferencias entre ambos métodos y se explican las ventajas e inconvenientes del uso de cada uno. Para finalizar, se comparan los dos métodos de forma práctica, al mostrar los resultados de clasificación que se obtienen cuando se aplican en una serie de dominios.

La característica principal que distingue los dos métodos es el momento de la ejecución en la que se sitúan los centros mediante el algoritmo K-Medias. En el primer método propuesto, los centros se sitúan al principio de la ejecución, utilizando el algoritmo K-Medias con la función de Distancia Euclídea. La posición de estos centros no varía durante todo el proceso de clasificación y optimización de las funciones de distancias. Por otro lado, en el método propuesto como variante del primero, los centros se vuelven a calcular cada vez que se determina el valor de adecuación de cada individuo al realizar la optimización de las funciones de distancia. La disposición de dichos centros se realiza mediante el algoritmo K-Medias utilizando las funciones de distancias correspondientes al individuo cuyo valor de adecuación se pretende calcular.

Para referirnos al primer o al segundo método, utilizaremos esta diferencia principal que influye en el cálculo del *fitness* de los individuos. En el primer caso el *fitness* se calcula manteniendo los *centroides fijos* y en el segundo se calcula tras situar los centros mediante *K-Medias*.

Respecto al cálculo del valor de adecuación de los individuos, hay otra característica que distingue a los dos métodos. Mientras que en el método de los *centroides fijos*, el valor de adecuación viene dado, simplemente, por el número de fallos cometidos en la clasificación; en el caso del segundo método se incluyen otros factores relativos a los valores de los elementos de la matriz. Estos valores se incluyeron para mejorar los resultados en los casos en los que el escalamiento de algunos atributos sea crítico, permitiendo discriminar atributos que pudieran ser irrelevantes para la clasificación.

Al utilizar K-Medias cada vez que se calcula el *fitness*, utilizando la función de distancia que se está evolucionando, en el segundo método sólo se emplean matrices con todos los valores no negativos, lo que hace menos probable que no se alcance la convergencia con K-Medias.

En cuanto al rendimiento de los algoritmos, lógicamente el hecho de ejecutar K-Medias cada vez que se calcula el valor de adecuación de un individuo repercute en el rendimiento, resultando el segundo método más lento.

Para comparar los algoritmos en cuanto a resultados y rendimiento, se han realizado una serie de experimentos que se resumen en las tablas 5.25, 5.26 y 5.27, que contienen los resultados de esta experimentación utilizando el algoritmo CMA-ES, estrategias evolutivas simples y estrategias evolutivas múltiples, respectivamente.

La configuración de los parámetros, para cada ejecución, se ha establecido atendiendo a los experimentos realizados previamente (véase la sección 5 *Experimentación*). Las características más destacables son las siguientes:

- Para el dominio 3 (*nubes rotadas*) se emplean matrices simétricas, para el resto se utilizan matrices diagonales.
- Para el dominio 5 (*atributos aleatorios*) se utiliza 10000 iteraciones máximas para el algoritmo CMA-ES, y 5000 para las estrategias evolutivas simples y múltiples, para las cuales se utiliza 0,5 como probabilidad de poner a cero un valor del cromosoma en cada iteración.

	Fitness centros fijos		**Fitness K-Medias**	
	Aciertos (%)	Tiempo (s.)	Aciertos (%)	Tiempo (s.)
Nubes alineadas	100.00	0.692	100.00	1.475
Nubes reflejo	100.00	0.794	82.00	10.794
Nubes rotadas	100.00	0.296	75.00	37.502
Elipses	99.62687	15.613	88.99254	1894.753
Atributos aleatorios	50.00	13.73	97.00	231.367

Tabla 5.25: Resultados de la comparativa con cma.

Al observar los resultados obtenidos llama la atención que, redistribuyendo los centros al calcular el *fitness* los resultados obtenidos para dominios como el 3 y el 4 (*nubes reflejo* y *nubes rotadas*), la tasa de aciertos obtenida no es tan elevada como al utilizar el primer método propuesto. Se ha de tener en cuenta que los experimentos se han realizado con el mismo número máximo de iteraciones para los dos métodos, y probablemente se alcanzarían mejores resultados con el segundo método si este número de iteraciones fuera mayor (al igual que ocurría en las pruebas explicadas en la sección 5.6.3). Además, para

	Fitness centros fijos		**Fitness K-Medias**	
	Aciertos (%)	Tiempo (s.)	Aciertos (%)	Tiempo (s.)
Nubes alineadas	100.00	0.297	100.00	0.977
Nubes reflejo	100.00	1.463	81.33	7.543
Nubes rotadas	100.00	0.17	50.25	6.488
Elipses	98.13433	20.928	71.45522	14462.962
Atributos aleatorios	47.00	5.639	100.00	2.12

Tabla 5.26: Resultados de la comparativa con ee1.

	Fitness centros fijos		**Fitness K-Medias**	
	Aciertos (%)	Tiempo (s.)	Aciertos (%)	Tiempo (s.)
Nubes alineadas	100.00	0.953	100.00	2.712
Nubes reflejo	100.00	2.998	60.0	23.379
Nubes rotadas	100.00	0.553	73.25	379.453
Elipses	96.64179	715.311	—	—
Atributos aleatorios	51.00	221.367	100.00	15.807

Tabla 5.27: Resultados de la comparativa con eem.

el dominio *nubes rotadas* se está empleando el valor absoluto de las matrices en el caso del segundo método, resultando aún más complicado obtener una función de distancia óptima, tal como se pudo observar en los experimentos presentados en la sección 5.3, *Dominio 3: nubes rotadas*.

Para el dominio 5 (*atributos aleatorios*) sólo se obtienen buenos resultados utilizando el segundo método. Además, en las ejecuciones con técnicas EE-1+1 y estrategias evolutivas múltiples, la solución se alcanza muy rápidamente; lo cual es efecto del nuevo operador genético introducido, habiendo especificado 0,5 como probabilidad de poner a cero un valor del cromosoma en cada iteración.

Por otro lado, es fácil observar el hecho de que las soluciones al mantener los centros en la misma posición durante la optimización de la matriz de distancia se alcanzan mucho más rápido que si estos centros se redistribuyen cada vez que realiza la clasificación. Esto se acentúa especialmente en el dominio 4 (*elipses*), que es el dominio con mayor número de datos: pasa de alrededor de 15 segundos a unos 32 minutos en la ejecución con el algoritmo CMA-ES y de casi 21 segundos a unas 4 horas en el caso al utilizar EE-1+1. La ejecución con el segundo método y las estrategias evolutivas múltiples se interrumpió cuando tras

más de 12 horas de ejecución sólo se habían completado los tres primeros subconjuntos de los diez de validación cruzada.

Es importante resaltar también que, en las ejecuciones del segundo método en el dominio 4, se dieron algunos casos en los que K-Medias no había convergido tras 10000 iteraciones, con la consecuente penalización de los individuos correspondientes. Esto hecho muestra que esta situación no sólo se da cuando no se restringen los valores de las matrices. Un ejemplo de función con la que habría problemas de convergencia para el algoritmo K-Medias es la correspondiente a una matriz nula, pues la distancia entre dos puntos cualesquiera sería igual a cero.

Por lo tanto, se puede concluir la comparativa entre los dos métodos propuestos poniendo de manifiesto que aunque la variante sugerida introduce mecanismos que mejoran la ponderación de los distintos atributos y permite discriminar los que son irrelevantes más fácilmente, su uso conlleva un tiempo de ejecución mucho mayor; pudiéndose no alcanzar soluciones adecuadas para ciertos dominios a las que se llega rápidamente con el primer método, si no se utiliza un número suficiente de iteraciones.

Capítulo 6

Conclusiones

Este capítulo se ha dividido en dos partes: por una parte, se hace un repaso de los objetivos marcados y se comenta en qué medida se han visto satisfechos, y por otra parte, se resumen las conclusiones extraídas de la experimentación para explicar sucintamente las ventajas e inconvenientes de las técnicas clasificadoras que han sido objeto de estudio.

6.1. Objetivos alcanzados

Al desarrollar este proyecto se han cumplido, en la medida de lo posible, todos los objetivos que se habían establecido:

- Se ha definido, diseñado e implementado un método de clasificación basado en prototipos en el que se integra la optimización de la función de distancia mediante el uso de estrategias evolutivas.
- El funcionamiento y rendimiento de dicho método se ha podido medir comparando la tasa de aciertos en clasificación con la obtenida antes de optimizar la función de distancia, es decir, empleando la Distancia Euclídea.
- Se ha diseñado un conjunto de dominios de datos que han permitido comprender las propiedades que presenta el algoritmo y han permitido evaluarlo mediante la experimentación.
- A partir de dicha experimentación se han detectado limitaciones en el método propuesto (véase la sección 5.5). Como respuesta a estos resultados, se ha planteado e implementado una variante del método que no tuviera esas limitaciones.

6.2. Conclusiones finales

En la sección 5 *Experimentación*, se ha podido comprobar que la tasa de aciertos en clasificación se mejora, en gran medida, para los cuatro primeros dominios, tras optimizar la función de distancia, siempre que se halle la configuración de parámetros adecuada. Lo que quiere decir que este método resulta un buen clasificador para dominios de este tipo.

También se ha podido observar que no se podían obtener mejoras significativas respecto a la clasificación con la Distancia Euclídea en el quinto dominio planteado. Pues la disposición inicial de los centros mediante con K-Medias y la propia Distancia Euclídea está más influenciada por los atributos más significativos; y precisamente en este dominio, los atributos más significativos son irrelevantes a la hora de clasificar.

Mediante la variante propuesta para solventar este problema, se obtienen buenos resultados en el dominio con atributos aleatorios, por lo que los cambios introducidos, realmente han servido para superar la limitación que se había detectado.

Pero dichos cambios aumentan, en alto grado, la complejidad del algoritmo; aumentando, consecuentemente, el tiempo de ejecución necesario para alcanzar una solución. Por otro lado, se introduce un nuevo factor que influye negativamente: no se alcanza la convergencia del algoritmo K-Medias para las funciones de distancia correspondientes a todas las posibles matrices.

Por otro lado, entre los tipos de estrategias evolutivas con los que se ha experimentado, el algoritmo CMA-ES es la técnica de optimización con la que, generalmente se han obtenido mejores resultados. Con la técnica EE-(1+1) se han obtenido buenos resultados para algunos dominios, mientras que para otros ha sido posible apreciar que los otros dos métodos empleados presentan capacidades superiores respecto a la exploración de soluciones. Las estrategias evolutivas con poblaciones no presentan estas carencias en cuanto a la exploración de soluciones, pero han necesitado un tiempo de cómputo mucho mayor que el resto de las técnicas empleadas.

También es importante señalar que la experimentación llevada a cabo nos ha permitido comprender las particularidades de la optimización de las funciones de distancia. Se han podido observar propiedades interesantes al relajar algunas de las restricciones de estas funciones, a la vez que se ha podido comprobar que resulta totalmente necesario, a la par que costoso, complir todas las restricciones en algunos contextos.

Por último, destacar que el diseño de los dominios de datos y las pruebas realizadas con ellos nos ha permitido comprender experimentalmente y en profundidad las propiedades de las técnicas de clasificación definidas, así como el efecto de cada uno de los parámetros que es necesario determinar al emplearlas.

Capítulo 7

Líneas futuras de investigación

En este capítulo se describen algunas posibles líneas futuras de investigación que pueden llevarse a cabo tomando como base el trabajo realizado en este proyecto.

1. **Experimentación en dominios reales.**

   En este proyecto se ha propuesto un método de clasificación basado en optimizar la función de distancia para un clasificador que emplea prototipos. Mediante la generación de una serie de dominios artificiales ha sido posible comprender el funcionamiento del mismo y determinar las ventajas e inconvenientes de emplearlo frente a distintos tipos de dominios. Una posible extensión de este proyecto sería la experimentación con dominios reales, pudiendo comparar los resultados con los que se obtendrían utilizando otras técnicas de clasificación.

2. **Tratamiento o restricción de las "distancias negativas".**

   Como se explicó en la sección 4.1.2 (en la página 34), existen problemas al tratar optimizar las matrices de distancia, dada la necesidad de que sean definidas positivas para que las funciones correspondientes cumplan todas las propiedades que caracterizan a una métrica de distancia.

   Este problema se ha intentado solucionar de dos formas: por una parte, se han considerado como infinitas todas las distancias cuyo cuadrado sea menor que cero, y por otra parte, se ha utilizado el valor absoluto de las matrices que se están optimizando (garantizando así que se trata de matrices con todos los valores no negativos, las cuales son un subconjunto de las matrices definidas positivas).

   La primera solución presenta propiedades interesantes para algunos dominios (se encuentran buenas soluciones en un tiempo de cómputo reducido), pero presenta

ciertos inconvenientes, como que resulta probable que K-Medias no converja utilizando este tipo de funciones de distancia.

En cuanto a la segunda solución, garantiza que las matrices de distancia optimizadas van a dar lugar a una función de distancia válida, pero además de que el uso del valor absoluto de las matrices obtenida introduce redundancias en la clasificación, hay matrices definidas positivas que quedan fuera de la representación; por lo que esas potenciales soluciones no se podrán alcanzar.

Por lo tanto, otra posible extensión del proyecto sería encontrar un mejor tratamiento de las "distancias negativas", o una forma eficiente de garantizar que las matrices obtenidas sean definidas positivas. Por ejemplo, podrían tratar de evolucionarse los autovalores de las matrices y obtener las matrices correspondientes.

3. **Codificación de los centros en el cromosoma.**

   En la sección 5.6 se propone distribuir los centros cada vez que se calcula el valor de adecuación de un individuo para poder discriminar atributos que no sean relevantes a la clasificación (la influencia de estos al calcular inicialmente los centros utilizando la Distancia Euclídea, provoca que en algunos casos no se puedan alcanzar soluciones satisfactorias). Esta solución propuesta afecta al rendimiento del algoritmo y hace imprescindible utilizar matrices que no contengan valores negativos, para evitar ejecuciones fallidas del algoritmo K-Medias.

   Se podrían codificar los centroides en el cromosoma, evolucionándolos junto a las matrices de distancia. Esto aumenta el espacio de búsqueda para el método de optimización, pero al no necesitar ejecutar K-Medias cada vez que se calcula el valor de adecuación de un individuo para redistribuir los centros, es posible que el tiempo de ejecución necesario se reduzca. Por otra parte, este mismo hecho, permite relajar las restricciones de las funciones de distancia (al no ejecutar K-Medias cada vez, no es preciso tratar de asegurar su convergencia), por lo que no es estrictamente necesario que las matrices sean definidas positivas, siempre que las "distancias negativas" se traten adecuadamente.

4. **Uso de otros métodos de clasificación y otros métodos de optimización de la función de distancia.**

   En este trabajo se han utilizado estrategias evolutivas para optimizar las matrices de distancia y un método de clasificación basado en K-Medias. Para optimizar las funciones de distancia podrían emplearse otras técnicas de optimización. En cuanto al método de clasificación, podrían emplearse otros métodos basados en prototipos.

Anexo A

Manual de usuario

En este manual se explica cómo utilizar la aplicación desarrollada en este proyecto, permitiendo al usuario realizar experimentos con las distintas opciones de configuración del algoritmo que ha sido propuesto.

Con este propósito se explican los siguientes aspectos:

- Requisitos mínimos
- Formato de los ficheros de datos
- Parámetros de configuración
- Ejecución de la aplicación
- Datos de salida
- Ejemplo sencillo de ejecución de la aplicación
- Generación de dominios de datos

Los ejecutables de la aplicación, el código fuente, la documentación del código, este manual de usuario, los dominios utilizados y una breve descripción del algoritmo pueden obtenerse en el siguiente sitio web: `http://www.lab.inf.uc3m.es/~christian/kmes`.

A.1. Requisitos mínimos

- Para ejecutar la aplicación se requiere tener instalado el entorno de ejecución Java (*JRE, Java Runtime Environment*) 1.6 o superior[1]. Se puede ejecutar la aplicación en cualquier sistema compatible con dicho entorno de ejecución, independientemente del Sistema Operativo.

- Al tratarse de una aplicación que se ejecutan en línea de comandos, no se precisa interfaz gráfica.

A.2. Formato de los ficheros de datos

Se utilizan ficheros de texto planto, estando los patrones separados por saltos de línea y cada uno de sus atributos por espacios y/o tabuladores. En la figura A.2 se representa un fichero con formato válido, para que el formato de los ficheros sea correcto deben cumplirse las siguientes condiciones:

- Todos los patrones del mismo dominio tienen que tener el mismo número de atributos y una clase, la cual se especifica a continuación de los atributos.

- Cada uno de estos atributos será un número real representable según el estándar IEEE 754. Utilizándose el punto como separador decimal y admitiéndose los valores en formato exponencial, por ejemplo: 1.61e-18 ó -2.15e+9.

- La clase consistirá en una cadena de caracteres que no contenga espacios ni tabuladores.

- El tamaño de los ficheros queda limitado por el hardware del sistema, ya que los datos se cargan en memoria.

[1]Se puede obtener en `http://java.sun.com`.

```
-0.109742769    0.696514698    0.52099256    clase1
-1.456e-11      0.869775036    0.54295162    clase1
2.51128e+10     0.76606136     0.70634576    clase2
-0.526626592    0.418687316    0.240570129   clase2
0.49226224      0.46876241     0.680851488   clase3
-0.064478127    0.83407012     0.778361218   clase3
-0.516451834    0.50125611     0.619578181   clase1
```

Figura A.1: Ejemplo de fichero de datos válido.

A.3. Parámetros de configuración

Todos los parámetros de la aplicación se configuran en un fichero de propiedades. En este fichero, se permite poner comentarios en el fichero, que serán ignorados por la aplicación, mediante el carácter #. Cada parámetro de configuración se especifica mediante uno o varios campos, los cuales constan de un nombre que precede al signo igual, tras el que se especifica el valor (*campo = valor*). En la tabla A.1 se detallan las propiedades generales de configuración.

Campo	**Descripción**	**Valores**
matriz.tipo	Permite indicar las características de las matrices que se van a emplear. Permite determinar dos tipos de matrices, diagonales y simétricas.	*diagonal, simétrica.*
matriz.por	Permite especificar si se utilizará una matriz por clase o por centroide.	*clase, centroide.*
matriz.signo	Permite determinar si se utilizarán matrices con sólo valores positivos o con cualquier valor real.	*positiva, general.* Opcional, valor por defecto: *general.*
validacion	Se determina el número de subconjuntos utilizados para realizar la validación cruzada.	Admite cualquier valor natural, debe ser menor que el número de datos a utilizar en la ejecución. Típicamente se emplea 10.

Campo	**Descripción**	**Valores**
centroides	Establece el número de centroides por clase a utilizar.	Dos posibilidades: - Un número natural n. Se utilizan n centroides por clase. - Una lista de números naturales separados por comas $(n_1, n_2, \ldots, n_k)$. n_1 para la primera clase, n_2 para la segunda.... El orden de las clases es el de aparición en el fichero de datos.
fitness	Permite determinar la técnica a emplear para calcular el valor de adecuación de los individuos.	Dos posibilidades: - *centroides-fijos*: se utiliza el primer método propuesto. Se sitúan los centros al principio con K-Medias y se mantienen durante la evolución. - *k-medias*: se emplea el segundo método propuesto. Se ejecuta K-Medias cada vez que se calcula

Campo	**Descripción**	**Valores**
inicializacion_k-medias	Determina cómo se inicializan los centroides al ejecutar el algoritmo K-Medias.	Tres posibilidades: - *aleatorio*: para cada centro utiliza un valor aleatorio de entre los patrones de datos. - *aleatorio-extremos*: cada centro se inicia con un aleatorio, cada atributo está en el rango definido por los máximos y mínimos de los patrones de datos. - *k-means++*: se utiliza el método K-Medias++.
convergencia	Permite establecer el criterio de convergencia para las ejecuciones de K-Medias.	Se permiten valores reales, se suelen utilizar valores muy próximos a cero.
algoritmos	Mediante esta propiedad se establecen los nombres de las técnicas a emplear para la optimización. Estas técnicas deberán tener una configuración específica.	Lista de cadenas de caracteres separadas por comas.

Tabla A.1: Propiedades generales de configuración.

Para todos los nombres de algoritmos que se hayan especificado en la campo *algoritmos*, se debe incluir la configuración específica de la técnica correspondiente. La configuración de cada técnica también se especifica mediante propiedades, en este caso se especifican indicando el nombre del algoritmo para cada característica específica, para ello se emplea "*nombre.campo = valor*".

La propiedad **tipo** es obligatoria para todas las técnicas que se especifiquen y admite tres posibles valores: *cma-es* para estrategias evolutivas de tipo CMA-ES, *ee1* para estrategias evolutivas de tipo EE-(1+1), y *eem* para estrategias evolutivas con poblaciones.

Según el tipo de estrategia evolutiva indicada, se deben especificar unos parámetros de configuración determinados. Es posible especificar varios algoritmos del mismo tipo, con distintas configuraciones. El único parámetro específico de los algoritmos obligatorio, es el tipo, para el resto de parámetros se emplean los valores por defecto, en caso de no especificarse.

Las tablas A.2, A.3 y A.4 detallan los parámetros de configuración de los tipos de algoritmo cma-es, ee1 y eem, respectivamente.

Campo	**Descripción**	**Valores**
cromosoma	Permite establecer los valores iniciales del cromosoma.	Admite valores reales, se puede emplear de dos formas: - Especificando un solo valor al que se inicializan todos los valores del cromosoma. - Especificando una lista de valores para inicializar cada elemento del cromosoma, el número de valores dependerá del tamaño del cromosoma. Por defecto: 0.1.

Campo	**Descripción**	**Valores**
varianza	Permite determinar los valores de la varianza inicial.	Los valores admitidos son equivalentes a la propiedad *cromosoma*. Por defecto: 1
umbralinicializacion	Utilizando este campo se puede aleatorizar la inicialización del cromosoma, para ello se establece un rango para utilizar valores aleatorios dentro de éste.	Los valores admitidos son equivalentes a la propiedad *cromosoma*. Por defecto: 0
generacionesmaximas	Establece el número máximo de iteraciones del algoritmo CMA-ES.	Un número natural. Por defecto: 1000

Tabla A.2: Propiedades para los algoritmos cma-es.

Campo	Descripción	Valores
cromosoma	Permite establecer los valores iniciales del cromosoma.	Admite valores reales, se puede emplear de dos formas: - Especificando un solo valor al que se inicializan todos los valores del cromosoma. - Especificando una lista de valores para inicializar cada elemento del cromosoma, el número de valores dependerá del tamaño del cromosoma. Por defecto: 0.1
varianza	Permite determinar los valores de la varianza inicial.	Los valores admitidos son equivalentes a la propiedad *cromosoma*. Por defecto: 1
umbralinicializacion	Utilizando este campo se puede aleatorizar la inicialización del cromosoma, para ello se establece un rango para utilizar valores aleatorios dentro de éste.	Los valores admitidos son equivalentes a la propiedad *cromosoma*. Por defecto: 0
generacionesmaximas	Establece el número máximo de iteraciones del algoritmo ES-1+1.	Un número natural. Por defecto: 1000
convergencia	Permite determinar el valor que tiene que alcanzar la varianza para asumir que se ha convergido a una solución.	Permite valores reales, generalmente se emplean valores muy cercanos a cero. Por defecto: 1e-5

Campo	**Descripción**	**Valores**
mutacion-cero	Permite especificar la probabilidad de que, en cada generación, uno de los valores del cromosoma mute a 0.	Admite valores reales entre cero y uno. Por defecto: 0

Tabla A.3: Propiedades para los algoritmos ee1.

Campo	Descripción	Valores
cromosoma	Permite establecer los valores iniciales del cromosoma.	Admite valores reales, se puede emplear de dos formas: - Especificando un solo valor al que se inicializan todos los valores del cromosoma. - Especificando una lista de valores para inicializar cada elemento del cromosoma, el número de valores dependerá del tamaño del cromosoma. Por defecto: 0.1
varianza	Permite determinar los valores de la varianza inicial.	Los valores admitidos son equivalentes a la propiedad *cromosoma.* Por defecto: 1
umbralinicializacion	Utilizando este campo se puede aleatorizar la inicialización del cromosoma, para ello se establece un rango para utilizar valores aleatorios dentro de éste.	Los valores admitidos son equivalentes a la propiedad *cromosoma.* Por defecto: 0
generacionesmaximas	Establece el número máximo de iteraciones de la estrategia evolutiva múltiple.	Admite números naturales. Por defecto: 1000
convergencia	Permite determinar el valor que tiene que alcanzar la varianza para asumir que se ha convergido a una solución.	Permite valores reales, generalmente se emplean valores muy cercanos a cero.

Campo	Descripción	Valores
poblacion	Se especifica el tamaño de la población de individuos.	Admite números naturales mayores que dos. Por defecto: 50
cruce	Especifica la proporción de la población que se reproducirá en cada generación.	Admite valores reales entre cero y uno. Si se utiliza 1 no se realiza torneo, ya que se cruzan todos los individuos. Por defecto: 0.5
torneo	Especifica la proporción de la población que se selecciona en cada ronda de los torneos.	Admite valores reales entre cero y uno. Por defecto: 0.1
mutacion-cero	Permite especificar la probabilidad de que, en cada generación, uno de los valores del cromosoma mute a 0.	Admite valores reales entre cero y uno. Por defecto: 0

Tabla A.4: Propiedades para los algoritmos eem.

A modo de ejemplo, a continuación, se presentan fragmentos de un fichero de configuración válido tanto para las opciones de configuración generales como para las específicas.

```
#matriz.tipo = diagonal
matriz.tipo = simetrica
#matriz.por = clase
matriz.por = centroide
#Opcional: matriz.signo = [general | positiva], general por defecto
#---------------------
# Validación cruzada
#---------------------
# validacion = N
# Número de grupos de patrones en la validación cruzada
#
validacion = 10
#Centroides por clase
#centroides = [N | n1,n2, ... ,nN] (Para N clases)
centroides = 1
#centroides = 2,4
#Tipo de fitness
#fitness = k-medias
fitness = centroides-fijos
#Inicialización de los centroides del K-Medias
inicializacion_k-medias = kmeans++
#inicializacion_k-medias = aleatorio
#inicializacion_k-medias = aleatorio-extremos
#Convergencia para el K-Medias
convergencia = 0.000001
#--------------------
# Algoritmos
#--------------------
# algoritmos = <algoritmo1>,<algoritmo2>,<algoritmo3>, ... , <algoritmoN>
# Lista de los algoritmos (nombres utilizados en el fichero)
algoritmos = cma,ee1,eem
```

Figura A.2: Ejemplo de configuración general.

```
#CMA-ES
#--------------------
# Todos los parámetros salvo el tipo son opcionales, en caso de
# no especificarse se utilizan valores por defecto
# nombre.tipo = cma-es
# nombre.cromosoma = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.varianza = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.umbralinicializacion = [valor | valor1,valor2, ... , valor]
# (valor que establece el umbral en torno a nombre.cromosoma
# para la inicialización)
#
cma.tipo = cma-es
cma.cromosoma = 1.0
#cma.cromosoma = 0.5,1.8
cma.varianza = 1.0
#cma.varianza = 1.0,1.8
cma.umbralinicializacion = 0.5
#cma.umbralinicializacion = 1.5,0.5
cma.generacionesmaximas = 1000
```

Figura A.3: Ejemplo de configuración cma-es.

```
#EE(1+1)
#---------------------
# Todos los parámetros salvo el tipo son opcionales, en caso de
# no especificarse se utilizan valores por defecto
# nombre.tipo = ee1
# nombre.cromosoma = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.varianza = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.umbralinicializacion = [valor | valor1,valor2, ... , valor]
# (valor que establece el umbral en torno a nombre.cromosoma
# para la inicialización)
# nombre.generacionexMaximas = valor (Máximo de generaciones)
# nombre.convergencia = valor (diferencia mínima entre varianzas
# para determinar que se ha convergido)
ee1.tipo = ee-1
ee1.cromosoma = 1.0
#ee1.cromosoma = 0.5,1.8
ee1.varianza = 100.0
#ee1.varianza = 1.0,1.8
ee1.umbralinicializacion = 0.5
#ee1.umbralinicializacion = 1.5,0.5
ee1.generacionesmaximas = 1000
ee1.convergencia = 0.00000001
# Probabilidad de que cada elemento del cromosoma mute a
# cero en la mutación.
ee1.mutacion-cero = 0.1
```

Figura A.4: Ejemplo de configuración ee1.

```
#EE(Múltiple)
#---------------------
# Todos los parámetros salvo el tipo son opcionales, en caso de
# no especificarse se utilizan valores por defecto
# nombre.tipo = eem
# nombre.cromosoma = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.varianza = [valor | valor1,valor2, ... , valorN]
# (Se especifica un valor inicial, o un vector inicial)
# nombre.umbralinicializacion = [valor | valor1,valor2, ... , valor]
# (valor que establece el umbral en torno a nombre.cromosoma
# para la inicialización)
# nombre.generacionexMaximas = valor (Máximo de generaciones)
# alg3.poblacion = valor (tamaño de la población)
# alg3.cruce = valor (proporción de la población que se reproduce
# en cada ronda)
# alg3.torneo = valor (proporción de la población que compite
# en el torneo)
eem.tipo = ee-m
eem.cromosoma = 1.0
#eem.cromosoma = 0.5,1.8
eem.varianza = 100.0
#eem.varianza = 1.0,1.8
eem.umbralinicializacion = 0.5
#eem.umbralinicializacion = 1.5,0.5
eem.generacionesmaximas = 1000
eem.convergencia = 0.00000001
eem.poblacion = 20
eem.cruce = 0.4
eem.torneo = 0.2
# Probabilidad de que cada elemento del cromosoma mute a
# cero en la mutación.
eem.mutacion-cero = 0.1
```

Figura A.5: Ejemplo de configuración eem.

A.4. Ejecución de la aplicación

En la figura A.6 se especifica cómo ejecutar la aplicación. Se especifican los siguientes parámetros:

- **CONFIG**: fichero en el que están especificados los parámetros de configuración.
- **DATOS**: fichero que contiene los datos para realizar la ejecución.
- **m**: si se pasa este parámetro, tras finalizar la validación cruzada, se realizará una última ejecución utilizando todos los datos, con el fin de obtener las matrices de distancia.

```
java -jar kmes.jar CONFIG DATOS [m]
```

Figura A.6: Ejecución de la aplicación.

Si no se indican los parámetros obligatorios, se muestra una pequeña ayuda indicando cómo han de suministrarse los mismos.

A.5. Datos de salida

Durante la ejecución del programa se va mostrando información sobre la validación cruzada por la salida estándar de error (*stderror*). Se indica de cuántos subconjuntos consta la validación cruzada y se indica cuándo se comienza y finaliza la ejecución de cada subconjunto.

Una vez finalizada la ejecución, se muestra la siguiente información por la salida estándar (*stdout*):

- La fecha y la hora de la ejecución.
- El número de técnicas utilizadas para optimizar la matriz de distancias.
- El número de patrones de datos de cada clase.

- El número de subconjuntos empleados en la validación cruzada.
- Los resultados de la validación cruzada, para ello se muestra la siguiente información para cada subconjunto:
  - Identificador del subconjunto.
  - Porcentaje de aciertos utilizando la Distancia Euclídea.
  - Porcentaje de aciertos de entrenamiento y validación para cada algoritmo.
  - Tiempo de ejecución para cada algoritmo, en segundos.
- Un resumen de los resultados, en el que se muestra el porcentaje de aciertos de validación medio para la Distancia Euclídea y cada uno de los algoritmos, así como el tiempo total para cada uno de los algoritmos.

Si además se especifica la opción *m* para realizar el cálculo de las matrices de distancias que se hayan hallado, también se muestra la siguiente información:

- Los centros calculados mediante K-Medias (tanto con la Distancia Euclídea como con las matrices calculadas).
- Las matrices obtenidas mediante cada algoritmo.
- La correspondencia entre las matrices y los centros calculados.

A.6. Ejemplo sencillo

En esta sección se explica cómo ejecutar la aplicación para un dominio de datos y una configuración determinada. Como configuración se va a emplear la determinada en las figuras A.2 y A.3. Esta configuración se incluye en el fichero *example.properties*, la única diferencia es que en lugar de determinar tres algoritmos como en el caso de la figura A.2, únicamente se emplea uno, tal como se muestra a continuación:

```
algoritmos=cma
```

Como dominio se utiliza *nubes-alineadas* (disponible en el fichero *nubes-alineadas.data*). Que tiene dos clases y 200 patrones de cada clase, con dos atributos.

La ejecución se realiza según lo especificado en la figura A.6, se especifica la opción *m* para calcular las matrices de distancia utilizando todos los datos.

```
java -jar kmes.jar example.properties nubes-alineadas.data m
```

Figura A.7: Ejemplo de llamada a la aplicación

Tras ejecutar este comando, se obtiene la salida indicada en las figuras A.8, A.9 y A.10.

```
Validación cruzada con 10 subconjuntos
Subconjunto 0... Hecho.
Subconjunto 1... Hecho.
Subconjunto 2... Hecho.
Subconjunto 3... Hecho.
Subconjunto 4... Hecho.
Subconjunto 5... Hecho.
Subconjunto 6... Hecho.
Subconjunto 7... Hecho.
Subconjunto 8... Hecho.
Subconjunto 9... Hecho.
Realizando entrenamiento con todos los datos para obtener las matrices...
Hecho.
```

Figura A.8: Datos mostrados por la salida de error

En la figura A.8 se muestran los datos obtenidos por la salida de error, estos mensajes se van mostrando según se va ejecutando cada subconjunto de validación cruzada, para informar al usuario de en qué punto de la ejecución se encuentra la aplicación.

La figura A.9 incluye toda la información relativa a la ejecución que se ha realizado. Primero incluye características del dominio de datos. Después muestra los centroides calculados mediante K-Medias con la Distancia Euclídea. A continuación, para cada algoritmo, muestra las matrices y los centroides, así como la correspondencia entre matrices (y funciones de distancia correspondientes) y los centroides; en este caso sólo se ha especificado un algoritmo, por lo que sólo aparecen los datos relativos al algoritmo *cma*.

```
# Fecha de ejecución: Tue Apr 13 20:16:56 CEST 2010
# Número de algoritmos utilizados: 1
# Conjunto de datos: 400 patrones.
#         - A: 200
#         - B: 200
# Validación cruzada con 10 subconjuntos.
#
# Centroides obtenidos con K-Medias:
#
# Clase A:
c0 =        [-2.0043275467803277, 0.017383201801843058]
#
# Clase B:
c1 =        [4.009279227605778, 2.977424265419081]
#
# Matrices obtenidas mediante las estrategias evolutivas:
#
# Matrices del cma:
#
m0 =         [3.118348878848902, 0.0;
         0.0, 1.9538109473855403]

m1 =         [0.03176167893239967, 0.0;
         0.0, -2.857382860501145]


#
# Centroides del cma:
#
c0 = [-2.0043275467803277, 0.017383201801843058]
c1 = [4.009279227605778, 2.977424265419081]
#
#
# Correspondencia entre centroides y matrices:
# c0 -> m0, c1 -> m1.
#
```

Figura A.9: Datos sobre la ejecución

En esta ejecución en concreto, se ha suministrado *fitness=centroides-fijos*, y por esta causa, los centroides del algoritmo *cma* tienen el mismo valor que los centroides obtenidos inicialmente con K-Medias. Si se hubiera utilizado *fitness=k-medias*, probablemente estos valores no coincidirían.

```
# Resultados de la validación cruzada:
# Subconjunto    Euclidea-train    Euclidea-test    cma-train(%)
cma-test(%)        Tiempo_cma_(s.)
0    50.00000    50.00000    100.00000    100.00000    0.576
1    50.00000    50.00000    100.00000    100.00000    0.133
2    50.00000    50.00000    100.00000    100.00000    0.051
3    50.00000    50.00000    100.00000    100.00000    0.175
4    50.00000    50.00000    100.00000    100.00000    0.133
5    50.00000    50.00000    100.00000    100.00000    0.09
6    50.00000    50.00000    100.00000    100.00000    0.09
7    50.00000    50.00000    100.00000    100.00000    0.049
8    50.00000    50.00000    100.00000    100.00000    0.09
9    50.00000    50.00000    100.00000    100.00000    0.132
#
# Resultados totales
# Algoritmo        Aciertos(%)        Tiempo_ejecucion(s.)
Euclidea        50.00
cma        100.00        1.519
```

Figura A.10: Resultados de la validación cruzada

Los datos de la figura A.10 muestran los resultados de la validación cruzada. En la primera columna se encuentra un identificador para cada subconjunto de la validación cruzada. Las dos siguientes corresponden a los porcentajes de aciertos para la Distancia Euclídea. A su derecha se encuentran dos columnas que indican los porcentajes de aciertos en entrenamiento y validación para al algoritmo *cma*; tras la cual se expresa, en segundos, el tiempo de ejecución que se ha requerido para los subconjuntos. Si se hubieran especificado más algoritmos, se incluirían tres columnas más por cada uno de ellos, indicando los porcentajes de aciertos y tiempo de ejecución correspondientes.

Bajo los resultados de la validación cruzada, se encuentra un resumen de los mismos, en el que se incluye la media de aciertos de validación para cada algoritmo y el tiempo de ejecución total.

A.7. Generación de dominios de datos

La aplicación incluye una utilidad integrada que permite generar distintos dominios de datos y realizar algunas operaciones con ellos.

Si se ejecuta el comando especificado en la figura A.11, se mostrará la ayuda del programa que muestra las posibles opciones que se encuentran en la figura A.12.

```
java -cp kmes.jar GenerarDatos help
```

Figura A.11: Ejecución del programa generador de dominios.

```
Uso: GenerarDatos <nombre> [opciones] | help

<nombre>:

nubes-alineadas
        Dominio simple. 100 datos, 50 de la clase A, 50 de la clase B

nubes-reflejo
        Dominio para dos centros por clase y una matriz por centro. 300
        datos, 150 de la clase rojo, 150 de la clase azul.

nubes-alineadas-rotar <n>
        Dominio equivalente nubes-alineadas pero rotado <n> grados.

random
        Dominio con cuatro atributos x1,x2,x3,x4. Los dos últimos atributos
        son aleatorios entre 0 y 100; x1 y x2 están entre 0 y 1. Si x1 > x2
        cada dato es de una clase, en caso contrario de la otra.

elipses
        Datos distribuidos de forma elíptica. 536 datos, 256 de cada clase.

mezclar <fichero>
        Ordena aleatoriamente los datos del fichero introducido y los muestra
        por la salida estándar.
```

Figura A.12: Ayuda del programa generador de dominios.

Bibliografía

[Arthur, 2007] David Arthur, Sergei Vassilvitskii. *K-means++: The advantages of careful seeding*. ACM-SIAM Symposium on Discrete Algorithms, 2007.

[Atkeson, 1997] Christopher G. Atkeson, Andrew W. Moore, Stefan Schaal. *Locally Weighted Learning*. Artificial Intelligence Review, Springer Netherlands, 1997.

[Bäck, 1991] Thomas Bäck, Frank Hoffmeister, Hans-Paul Schwefel. *A Survey of Evolution Strategies*. Proceedings of the Fourth International Conference on Genetic Algorithms (2-9), 1991.

[Darwin, 1921] Charles Darwin. *El origen de las especies*. Espasa Calpe, 1ª Edición, 1921. (Recurso electrónico disponible en la Biblioteca de Traductores: `http://www.traduccionliteraria.org/biblib/index.htm`).

[Dawkins, 2002] Richard Dawkins. *El Gen Egoísta. Las bases biológicas de nuestra conducta.* (134 - 136). Salvat Ciencia, 11ª Edición, 2002.

[Fogel, 1964] Fogel L.J., Owens A.J., Walsh M.J. *On the Evolution of Artificial Intelligence*. Proc. of the Fifth IEEE Natl. Symp. on Human Factors in Electronics, 1964.

[Gosling, 1996] James Gosling, Henry McGilton. *A White Paper. The Java Language Environment*. May, 1996. (Recurso disponible en Oracle Sun Developer Network: `http://java.sun.com/docs/white/langenv/`).

[Haasdonk, 2008] Bernard Haasdonk, Elzbieta Pekalska. *Classifcation with Kernel Mahalanobis Distance Classifiers*. Conference, Advances in Data Analysis, Data Handling and Business Intelligence, 2008.

[Hansen, 1996] Hansen, N. and A. Ostermeier. *Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation* In Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 312-317.

[Hansen, 2001] Hansen, N. and A. Ostermeier. *Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation.* Massachusets Institute of Technology. 9(2), pp. 159-195, 1994.

[Holland, 1975] Jonh Henry Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[Isasi, 2004] Pedro Isasi, Inés Galván. *Redes de neuronas artificiales. Un enfoque práctico.* Pearson Educación, 2004.

[Johnsonbaugh, 2002] Richard Johnsonbaugh, W. E. Pfaffenberger. *Foundations of Mathematical Analysis.* Dover Publications, 2002.

[Kohonen, 1986] Kohonen T. *Learning Vector Quantization.* Thechnical report, Helsinki Univ. of Tech., Otaniemi, 1986.

[Kohonen, 1995] Kohonen T. *Self-Organizing Maps.* Springer, Berlin, Heidelberg, 1995.

[Lindholm, 1999] Tim Lindholm, Frank Yellin. *The Java^{TM} Virtual Machine Specification. Second Edition.* 1999. (Recurso disponible en Oracle Sun Developer Network: `http://java.sun.com/docs/books/jvms/`).

[Lloyd, 1982] Stuart P. Lloyd. *Least Square Quantization in PCM* IEEE Transations on Information Theory, Vol. IT-28, N^{o} 2, March 1982.

[Mackay, 2008] MacKay, David. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2008. (Chapter 20. An Example Inference Task: Clustering).

[MacQueen, 1967] MacQueen, J. B. *Some Methods for classification and Analysis of Multivariate Observations.* Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 1967.

[Mahalanobis, 1936] Mahalanobis, P.C. *On the generalised distance in statistics.* Proceedings of the National Institute of Sciences of India, 1936.

[Meyer, 2000] Carl Dean Meyer. *Matrix analysis and applied linear algebra.* Society for Industrial and Applied Mathematics, 2000.

[Miller, 1995] Brad L. Miller, David E. Goldberg. *Genetic Algorithms, Tournament Selection, and the Effects of Noise.* IlliGAL Report. N^{o}95006 (1995).

[Mitchell, 1996] Melanie Mitchell. *An introduction to genetic algorithms.* Cambridge, MIT Press, 1996.

[Ozer, 2008] Sedat Ozer, Chi Hau Chen. *Generalized Chebyshev Kernels for Support Vector Classification.* ICPR 2008: 1-4.

[Rechenberg, 1973] Ingo Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[Schwefel, 1975] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. Dissertation, Technische Universität Berlin, May 1975.

[Valls, 2009] José M. Valls, Ricardo Aler. *Optimizing Linear and Quadratic Data Transformations for Classification Tasks*. International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2009). Burgos, 2009.

[Wang, 2004] Huei-Chun Wang, Chih-Chou Chiu, Chao-Ton Su. *Clasification Using the Mahalanobis-Taguchi System*. Journal of the Chinese Institute of Industrial Engineers, Vol. 21, No. 6, pp. 606-618 (2004).

[Wook-Ahn, 2006] Chang Wook Ahn. *Advances in Evolutionary Algorithms. Theory, Design and Practice*. Studies in Computational Intelligence, Vol. 18. Springer, 2006.